

1 Teresa M. Corbin (SBN 132360)
Denise M. De Mory (SBN 168076)
2 Jaclyn C. Fink (SBN 217913)
HOWREY LLP
3 525 Market Street, Suite 3600
San Francisco, California 94105
4 Telephone: (415) 848-4900
Facsimile: (415) 848-4999
5

Attorneys for Plaintiff SYNOPSYS and
6 Defendants AEROFLEX INCORPORATED,
AEROFLEX COLORADO SPRINGS, INC.,
7 AMI SEMICONDUCTOR, INC., MATROX
ELECTRONIC SYSTEMS, LTD., MATROX
8 GRAPHICS INC., MATROX
INTERNATIONAL CORP., and MATROX
9 TECH, INC.

10 UNITED STATES DISTRICT COURT
11 NORTHERN DISTRICT OF CALIFORNIA
12 SAN FRANCISCO DIVISION

13 RICOH COMPANY, LTD.,

14 Plaintiff,

15 vs.

16 AEROFLEX INCORPORATED, AMI
SEMICONDUCTOR, INC., MATROX
17 ELECTRONIC SYSTEMS LTD., MATROX
GRAPHICS INC., MATROX
18 INTERNATIONAL CORP., MATROX TECH,
INC., AND AEROFLEX COLORADO
19 SPRINGS, INC.,

20 Defendants.

21 SYNOPSYS, INC.,

22 Plaintiff,

23 vs.

24 RICOH COMPANY, LTD.,

25 Defendant.
26

Case No. C03-4669 MJJ (EMC)

Case No. C03-2289 MJJ (EMC)

**DECLARATION OF DENISE M. DE MORY
IN SUPPORT OF MOTIONS FOR
SUMMARY JUDGMENT**

1 I, Denise M. De Mory, declare as follows:

2 1. I am a partner at the law firm of Howrey LLP, counsel for Aeroflex Incorporated,
3 Aeroflex Colorado Springs, AMI Semiconductor, Inc., Matrox Electronic Systems, Ltd., Matrox
4 Graphics Inc., Matrox International Corp., and Matrox Tech, Inc. and Synopsys, Inc. (collectively, the
5 "Defendants") in this action. The following declaration is based on my personal knowledge. If called
6 upon to testify, I could and would competently testify to the matters set forth below.

7 2. Attached as Exhibit 1 is a true and correct copy of United States Patent No. 4,922,432.

8 3. Attached as Exhibit 2 is a true and correct copy of Ricoh's Patent Final Contentions
9 Pursuant to Patent L.R. 3-6 served March 24, 2006.

10 4. Attached as Exhibit 3 is a true and correct copy of Ricoh's Supplemental Patent Final
11 Contentions Pursuant to Patent L.R. 3-6 served June 23, 2006. [FILED UNDER SEAL]

12 5. Attached as Exhibit 4 is a true and correct copy of Exhibit 65 to Ricoh's Supplemental
13 Patent Final Contentions Pursuant to Patent L.R. 3-6 served June 23, 2006.

14 6. Attached as Exhibit 5 is a true and correct copy of Ricoh's Claim Construction Opening
15 Brief served on August 27, 2004.

16 7. Attached as Exhibit 6 is a true and correct copy of Responsive Claim Construction Brief
17 for U.S. Patent No. 4,922,432 (Re-Filed) served on September 14, 2004.

18 8. Attached as Exhibit 7 is a true and correct copy of Ricoh's Claim Construction Reply
19 Brief served on September 20, 2004.

20 9. Attached as Exhibit 8 is a true and correct copy of April 7, 2005 Claim Construction
21 Order.

22 10. Attached as Exhibit 9 are true and correct copies of Ricoh's Expert Report of Mario
23 Papaefthymiou on Infringement by Aeroflex (Ex. 9A), AMI (Ex. 9B) and Matrox (Ex. 9C) served June
24 23, 2006. [FILED UNDER SEAL]

25 11. Attached as Exhibit 10 are true and correct copy of Expert Deposition of Mario
26 Papeafthymiou dated August 11, 2006. [FILED UNDER SEAL]

1 12. Attached as Exhibit 11 are true and correct copies of Ricoh's Expert Report of Donald
2 Soderman on Infringement by Aeroflex (Ex. 11A), AMI (Ex. 11B) and Matrox (Ex. 11C). [FILED
3 UNDER SEAL]

4 13. Attached as Exhibit 12 are true and correct copy of Exhibit 1 to Ricoh's Expert Report
5 of Donald Soderman on Infringement by Aeroflex, AMI and Matrox. [FILED UNDER SEAL]

6 14. Attached as Exhibit 13 are true and correct copy of Expert Deposition of Donald
7 Soderman dated August 14 & 15, 2006. [FILED UNDER SEAL]

8 15. Attached as Exhibit 14 are true and correct copy of Ricoh's Supplemental Responses to
9 ASIC Defendants' Requests for Admissions.

10 16. Attached as Exhibit 15 are true and correct copy of April 26, 1989 Amendment to
11 Prosecution History for U.S. Patent No. 4,922,432.

12 17. Attached as Exhibit 16 are true and correct copy of November 24, 1989 Amendment to
13 Prosecution History for U.S. Patent No. 4,922,432.

14 18. Attached as Exhibit 17 are true and correct copy of United States Patent No. 6,226,776.

15 19. Attached as Exhibit 18 are true and correct copy of Article titled "Implementing 'C'
16 Designs in Hardware : A Full-Featured ANSI C to RTL Verilog Compiler in Action" by Donald
17 Soderman.

18 20. Attached as Exhibit 19 are true and correct copy of Article titled "Implementing C
19 Designs in Hardware: A Full-Featured ANSI C to RTL Verilog Compiler in Action" by Donald
20 Soderman.

21 21. Attached as Exhibit 20 are true and correct copy of Article titled "Implementing C
22 Alogrithms in Reconfigurable Hardware using C2Verilog" by Donald Soderman.

23 22. Attached as Exhibit 21 are true and correct copy of Book titled "Introduction to HDL-
24 Based Design Using VHDL" by Steve Carlson bearing bates numbers 2SP 0768299-2SP 0768485.

25 23. Attached as Exhibit 22 are true and correct copy of United States Patent No. 4,703,435.

26 24. Attached as Exhibit 23 are true and correct copy of Deposition of Yoon-Pin Simon Foo
27 dated May 31, 2006. [FILED UNDER SEAL]

1 25. Attached as Exhibit 24 are true and correct copy of master Thesis by Yoon-Pin Simon
2 Foo titled "Managing VLSI D Design Data with A Relational Database System" bearing bates
3 numbers FOO 000038-FOO 000156.

4 26. Attached as Exhibit 25 are true and correct copy of University of South Carolina Class
5 Enrollment Sheet for Topics/Computer Engr bearing bates number SC 0003406.

6 27. Attached as Exhibit 26 are true and correct copy of Project Description for ECE890B
7 Spring 1986 titled "A Knowledge-based Behavioral Description Translator" bearing bates number
8 FOO 000189-FOO 000191 and Foo Deposition Exhibit No. 521.

9 28. Attached as Exhibit 27 are true and correct copy of Article titled "A Framework for
10 Managing VLSI CAD Data" by Yoon-Pin Foo and H. Kobayashi bearing bates numbers
11 KBSC000904-KBSC000913 and Foo Deposition Exhibit No. 509.

12 29. Attached as Exhibit 28 are true and correct copy of Article titled "A Knowledge Based
13 System for VLSI Module Selection" by Yoon-Pin Foo and H. Kobayashi bearing bates numbers
14 KBSC000904-KBSC000913 and Foo Deposition Exhibit No. 508.

15 30. Attached as Exhibit 29 are true and correct copy of Deposition Transcript of Hideaki
16 Kobayashi dated May 25, 2006. [FILED UNDER SEAL]

17 31. Attached as Exhibit 30 are true and correct copy of document bearing bates numbers
18 KBSC00009-KBSC00028. [FILED UNDER SEAL]

19 32. Attached as Exhibit 31 are true and correct copies of Translation of bates numbered
20 page RCL0011957B. [FILED UNDER SEAL]

21 33. Attached as Exhibit 32 are true and correct copy of Translation of bates numbered page
22 RCL0011958B. [FILED UNDER SEAL]

23 34. Attached as Exhibit 33 are true and correct copy of document bearing bates number
24 RCL0011963. [FILED UNDER SEAL]

25 35. Attached as Exhibit 34 are true and correct copy of Exhibit A to the JCC Statement.

26 36. Attached as Exhibit 35 are true and correct copy of Master Thesis by Thaddeus J.
27 Kowalski titled "The VLSI Design Automation Assistant: A Knowledge-Based Expert System".
28

1 37. Attached as Exhibit 36 are true and correct copy of Article titled "The VLSI Design
2 Automation Assistant: From Algorithms to Silicon.

3 38. Attached as Exhibit 37 are true and correct copy of Deposition Transcript of Thaddeus
4 J. Kowalski dated May 23, 2006. [FILED UNDER SEAL]

5 39. Attached as Exhibit 38 are true and correct copy of Request for Reexamination of
6 United States Patent No. 4,922,432 to the United State Patent and Trademark Office dated January 17,
7 2006.

8 40. Attached as Exhibit 39 are true and correct copy of the United State Patent and
9 Trademark Office Order Granting Reexamination of United States Patent No. 4,922,432 dated
10 February 24, 2006

11 41. Attached as Exhibit 40 are true and correct copy of States Patent No. 4,922,432 File
12 History.

13 42. Attached as Exhibit 41 are true and correct copy of document bates numbered
14 RCL000064.

15 43. Attached as Exhibit 42 are true and correct copy of document bates numbered
16 RCL000186.

17 44. Attached as Exhibit 43 are true and correct copy of Article titled "KBSC: A Knowledge
18 Based Approach to Automatic Logic Synthesis" by H. Kobayashi.

19 45. Attached as Exhibit 44 are true and correct copy of document bates numbered
20 KBSC000870-KBSC000872

21 46. Attached as Exhibit 45 are true and correct copy of 37 C.F.R. 1.56(a).

22 47. Attached as Exhibit 46 are true and correct copy of document bates numbered
23 RCL00022.

24 48. Attached as Exhibit 47 are true and correct copy of document bates numbered
25 RCL000188.

26 49. Attached as Exhibit 48 are true and correct copy of document bates numbered
27 RCL000197.

1 50. Attached as Exhibit 49 are true and correct copy of 37 C.F.R. 1.97-1.99.

2 51. Attached as Exhibit 50 are true and correct copy of the corrected second supplemental
3 Smith product declaration.

4 52. Attached as Exhibit 51 are true and correct copy of Deposition Transcript of Cliff
5 Warren dated June 6, 2006. [FILED UNDER SEAL]

6 53. Attached as Exhibit 52 are true and correct copy of Deposition Transcript of David
7 Tran, dated December 16, 2005 and June 6, 2006. [FILED UNDER SEAL]

8 54. Attached as Exhibit 53 are true and correct copy of Deposition Transcript of Reed
9 Packer dated June 8, 2006. [FILED UNDER SEAL]

10 55. Attached as Exhibit 54 are true and correct copy of Expert Report of R. Fred Lipscomb.
11 [FILED UNDER SEAL]

12 56. Attached as Exhibit 55 are true and correct copy of Expert Report of Maureen S.
13 Loftus. [FILED UNDER SEAL]

14 57. Attached as Exhibit 56 are true and correct copy of Deposition Transcript of David
15 Chiappini, Vol. 1 dated February 23, 2006. [FILED UNDER SEAL]

16 58. Attached as Exhibit 57 are true and correct copy of Deposition Transcript of David
17 Chiappini, Vol. 3 dated June 7, 2006. [FILED UNDER SEAL]

18 59. Attached as Exhibit 58 are true and correct copy of Erik Olson Declaration in Support
19 of Motion for Summary Judgment of Noninfringement under 35 U.S.C. Section 271(g).

20 60. Attached as Exhibit 59 are true and correct copy of Michael Heynes Declaration in
21 Support of Motion for Summary Judgment of Noninfringement under 35 U.S.C. Section 271(g).

22 61 Attached as Exhibit 60 are true and correct copy of Second Supplemental Declaration of
23 David Chiappini of Matrox Graphics, Inc.

24 62. Attached as Exhibit 61 are true and correct copy of Second Supplemental Declaration of
25 Eric Boisvert of Matrox Electronic Systems.

26 63. Attached as Exhibit 62 are true and correct copy of document bearing bates numbers
27 MGI0688426-MGI0688481. [FILED UNDER SEAL]

1 64. Attached as Exhibit 63 are true and correct copy of Schedule 2.1.1 the the Wagner
2 Expert Report. [FILED UNDER SEAL]

3 65. Attached as Exhibit 64 are true and correct copy of Deposition Transcript of Eric
4 Boivert dated June 6, 2006. [FILED UNDER SEAL]

5 66. Aeroflex performs ASIC design work for United States Government sub-contractors
6 operating under a Government prime contract. Contracts executed for such work incorporate an
7 authorization and consent clause (Federal Authorization Clause 52.227-1) either in the purchase order
8 or contract between Aeroflex and the sub-contractor or in the U.S. Government prime contract under
9 which the design work is performed. At least \$4,804,851 of Aeroflex sales are pursuant to government
10 contracts incorporating an express authorization and consent clause.

11 67. Aeroflex internally classifies its government contracts with a prefix of "gv" before the
12 contract number. Some government prime contracts referenced on purchase orders for product sales
13 pursuant to contracts designated "gv" for products at issue likely contain an authorization and consent
14 clause. In some cases, the subcontractors are Sandia National Laboratory, Los Alamos National
15 Laboratory, and Boeing Satellite Systems, for example. At least one prime contract was not locatable
16 through a Freedom of Information Act request.

17 68. Attached as Exhibit 65 are true and correct copy of Complaint dated January 21, 2003.

18 69. Attached as Exhibit 66 are true and correct copy of document bearing bates numbers
19 KBSC001109-KBSC001117.

20 70. Attached as Exhibit 67 are true and correct copy of translations of bates number
21 2SP0708285. [FILED UNDER SEAL]

22 71. Attached as Exhibit 68 are true and correct copy of Chart of Synopsys-Ricoh Contract.

23 72. Attached as Exhibit 69 are true and correct copy of document bearing bates numbers
24 SP00001-00032. [FILED UNDER SEAL]

25 73. Attached as Exhibit 70 are true and correct copy of Verilog HDL Compiler Reference
26 Manual bearing bates numbers SP04436-SP04536. [FILED UNDER SEAL]

1 74. Attached as Exhibit 71 are true and correct copy of Product of the Year Awards
2 document bearing bates numbers SP04649-SP04651.

3 75. Attached as Exhibit 72 are true and correct copy of Synopsys ASIC Partnerships
4 bearing bates numbers SP04707-SP04727. [FILED UNDER SEAL]

5 76. Attached as Exhibit 73 are true and correct copy of Article by Ann Steffora "Avant!
6 Shakes Up Front-End Design" – Juniper CAE Software – Product Announcement, Electronic New,
7 June 21, 1999

8 77. Attached as Exhibit 74 are true and correct copy of Article by Ray Weiss "Hot Design
9 Comb", Electronic Engineering Times, May 14, 1990

10 78. Attached as Exhibit 75 are true and correct copy of Article "CAE Software Gould...,
11 Electronic News, July 1, 1991".

12 79. Attached as Exhibit 76 are true and correct copy of document bearing bates numbers
13 2SP0763439-2SP763460.

14 80. Attached as Exhibit 77 are true and correct copy of AMIS website print out.

15 81. Attached as Exhibit 78 are true and correct copy of AMIS website print out.

16 82. Attached as Exhibit 79 are true and correct copy of UTMC website print out.

17 83. Attached as Exhibit 80 are true and correct copy of Synopsys web site print out.

18 84. Attached as Exhibit 81 are true and correct copy of Synopsys web site print out.

19 85. Attached as Exhibit 82 are true and correct copy of Ricoh's Amended Complaint

20 86. Attached as Exhibit 83 are true and correct copy of document bearing bates number
21 MGI0033893-MGI0033908. [FILED UNDER SEAL]

22 87. Attached as Exhibit 84 are true and correct copy document bearing bates number
23 RCL011421-RCL011422. [FILED UNDER SEAL]

24 88. Attached as Exhibit 85 are true and correct copy document bates numbered SP0168741-
25 SP0168776. [FILED UNDER SEAL]

26 89. Attached as Exhibit 86 are true and correct copy document bates numbered SP0167847-
27 SP0167881. [FILED UNDER SEAL]

1 90. Attached as Exhibit 87 are true and correct copy document bates numbered SP0121654-
2 SP0121667. [FILED UNDER SEAL]

3 91. Attached as Exhibit 88 are true and correct copy document bates numbered
4 MGI0001490. [FILED UNDER SEAL]

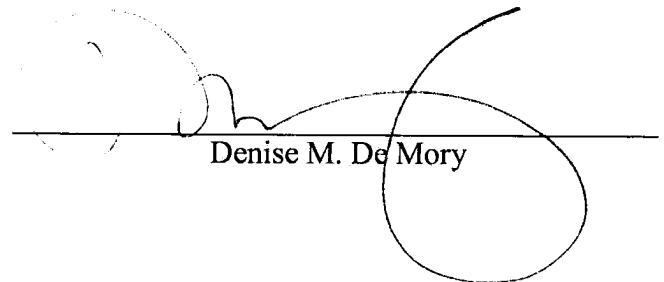
5 92. Attached as Exhibit 89 are true and correct copy Deposition Transcript of Shir-Shen
6 Chang dated January 5, 2006. [FILED UNDER SEAL]

7 93. Attached as Exhibit 90 are true and correct copy Deposition Transcript of Karen L.
8 Pieper dated December 12, 2005. [FILED UNDER SEAL]

9 Executed this 18th day of August, 2006, at San Francisco, California.

10 I declare under penalty of perjury under the laws of the United States of America that the
11 foregoing is true and correct.

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28



Denise M. De Mory

EXHIBIT

22

United States Patent [19]

Darringer et al.

[11] Patent Number: **4,703,435**[45] Date of Patent: **Oct. 27, 1987**[54] **LOGIC SYNTHESIZER**[75] Inventors: **John A. Darringer, Mahopac;**
William H. Joyner, Jr., Katonah,
both of N.Y.[73] Assignee: **International Business Machines**
Corporation, Armonk, N.Y.[21] Appl. No.: **631,364**[22] Filed: **Jul. 16, 1984**[51] Int. Cl.⁴ **G06F 7/00; G06F 15/60**[52] U.S. Cl. **364/489; 364/300;**
364/488[58] Field of Search **364/488, 489, 490, 491,**
364/300; 307/303[56] **References Cited****U.S. PATENT DOCUMENTS**4,377,849 3/1983 Finger et al. 364/491
4,580,228 4/1986 Noto 364/300 X
4,591,993 5/1986 Griffin et al. 364/491
4,612,618 9/1986 Pryor et al. 364/488 X**FOREIGN PATENT DOCUMENTS**

0168650 1/1986 European Pat. Off.

OTHER PUBLICATIONSFriedman et al., "Methods used in an Automatic Logic Design Generator (Alert)", *IEEE Trans. Comp. C-18*, pp. 593-614, 1969.Mitchell et al., "The Use of High Speed Proms to Generate Boolean Functions," *Wescon Technical Papers*, Sep. 1978, pp. 1-7.

Mano, "Digital Logic and Computer Design", Ch. 3, pp. 72-103, 1979.

Introduction to the Automated Synthesis of Computer; Herbert Schorr; Department of Electrical Engineering Digital Systems.

Laboratory Technical Report No. 16 (Mar. 1962). Ph.D. Thesis, Princeton University.

Minimization of Boolean Functions; F. J. Hill et al.; "Introduction to Switch Theory and Logical Design", 1973.

The Description, Simulation, and Automatic Imple-

mentation of Digital Computer Processors; John A. Darringer.

The Experimental Compiling System; F. E. Allen; IBM J. Res. Development; vol. 24, No. 6, Nov. 1980.

Logic Synthesis Through Local Transformation; John A. Darringer; IBM Journal of Research and Development; vol. 25, No. 4, Jul. 1981.

Programming Language; Yaohan Chu; vol. 8, No. 10, 10/65.

Development and Application of a Designer Oriented Cyclic Simulator; G. J. Parasch; 13th DA Conference 1976.

Logic Synthesis; Melvin A. Breuer; M. Breuer "Design Automation of Digital Systems", Prentice-Hall 1972.

Synthesis of Combinational Logic Networks; D. L. Dietmeyer "Logic Design of Digital Systems"; Allyn & Bacon, Boston 1978.

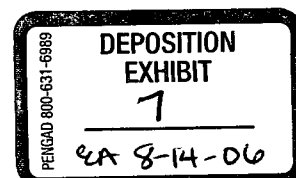
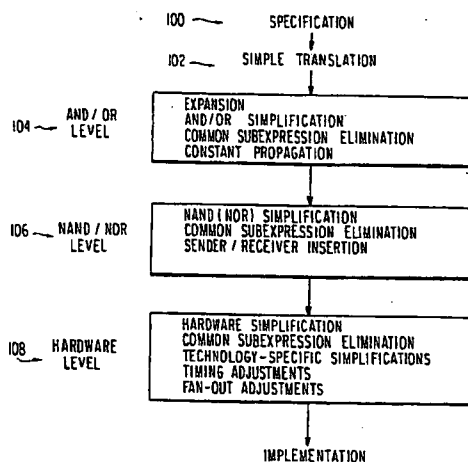
Quality of Designs from an Automatic Logic Generator (Alert)*; Theodore D. Friedman and Sih-Chin Yang; 7th DA Conference 1970.

On Logic Comparison; Leonard Berman; 18th DA Conference 1981.

(List continued on next page.)

Primary Examiner—Errol A. Krass*Assistant Examiner*—Joseph L. Dixon.*Attorney, Agent, or Firm*—Sughrue, Mion, Zinn, Macpeak, and Seas[57] **ABSTRACT**

Logic is synthesized from a flowchart-level description by first generating an AND/OR logic design, simplifying the AND/OR logic, converting the AND/OR logic to NAND or NOR logic, applying particular sequences of simplifying transformations to the NAND or NOR logic, converting the simplified NAND or NOR logic to a target technology, and simplifying the target technology where possible. The end result is an interconnection of primitives of the target technology in a language from which automated logic diagrams can be produced.

24 Claims, 33 Drawing Figures

4,703,435

Page 2

OTHER PUBLICATIONS

Automated Exploration of the Design Space for Register-Transfer (RT) Systems; Barbacci Mario Roberto; The Design and Analysis of an Automated Design Style Selector; Thomas Donald Earl, Jr.

Automation of Module Set Independent Register-Transfer Level Design; Edward Alfred Snow, III.

A new Look at Logic Synthesis; John A. Darringer; 17th DA Conference 1980.

Experiments in Logic Synthesis; John A. Darringer; IEEE ICCS 1980.

Methods used in an Automatic Logic Design Generator (Alert); Theodore D. Friedman; IEEE Transactions on Computers; vol. C-18, No. 7, Jul. 1969.

Register-Transfer Level Digital Design Automation: The Allocation Process; Louis Hafer; 15 DA Conference 1978.

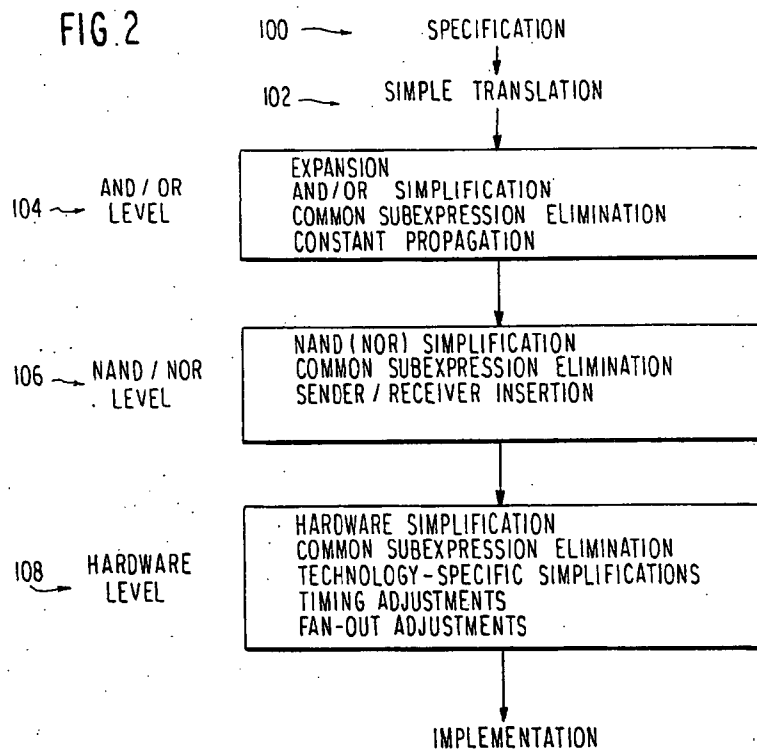
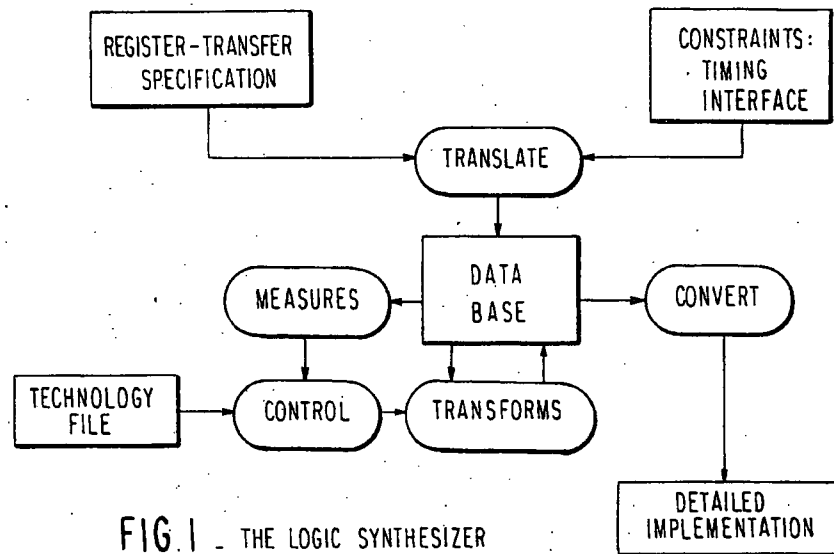
The CMU Design Automation System; An Example of Automated Data Path Design; A. Parker et al.; 16 DA Conference 1979.

Lores-Logic Reorganization System; Shunichiro Nakamura et al.; 15 DA Conference 1978.

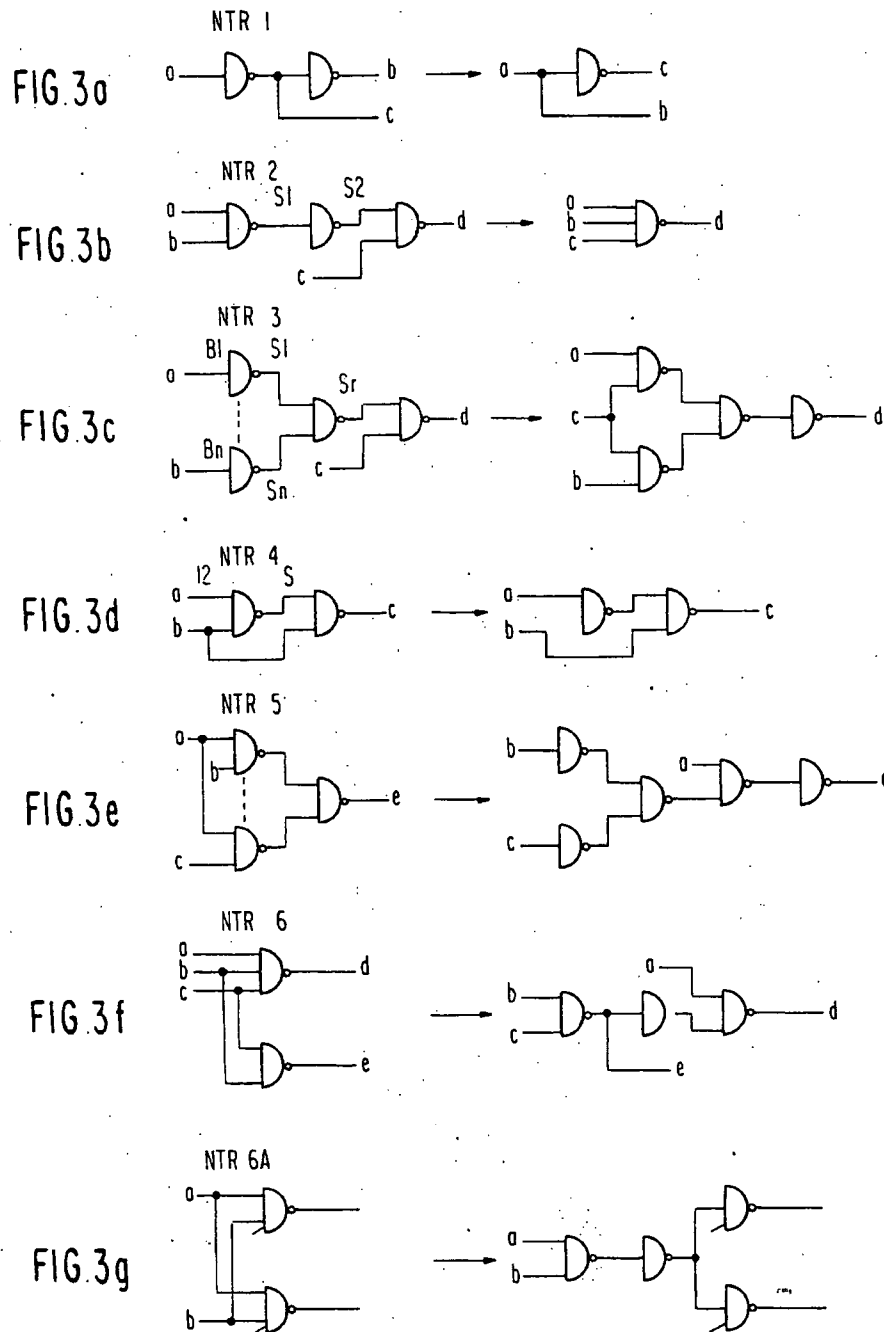
Translation of a DDL Digital System Specification to Boolean Equations; James R. Duley and Donald L. Dietmeyer, IEEE Trans. on Comp. vol. C-18, No. 14, '69.

DDL-A Digital System Design Language; James Robert Duley; Ph.D. 1967.

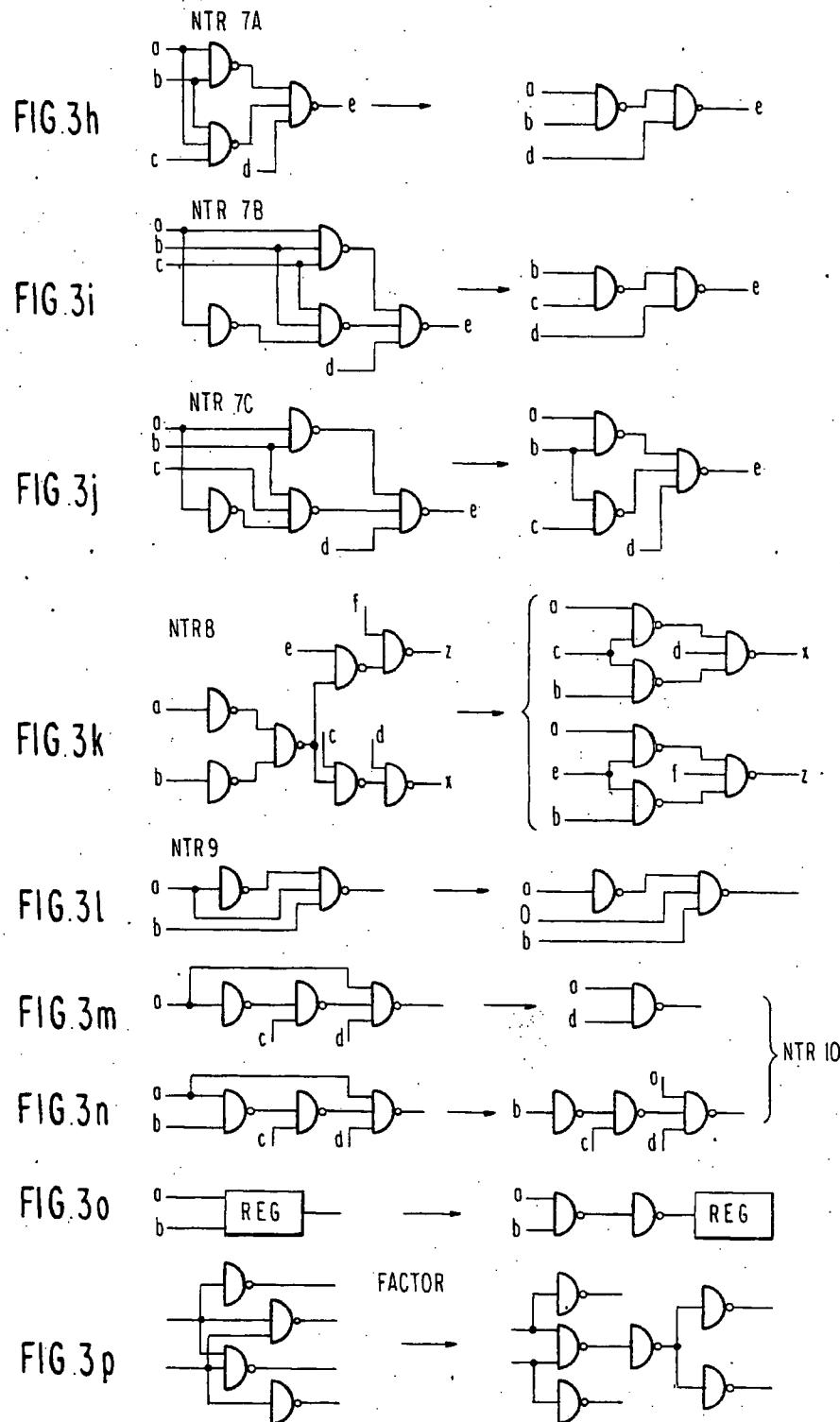
U.S. Patent Oct. 27, 1987 Sheet 1 of 5 4,703,435



U.S. Patent Oct. 27, 1987 Sheet 2 of 5 4,703,435



U.S. Patent **Oct. 27, 1987** **Sheet 3 of 5** **4,703,435**



U.S. Patent Oct. 27, 1987 Sheet 4 of 5 4,703,435

FIG. 4

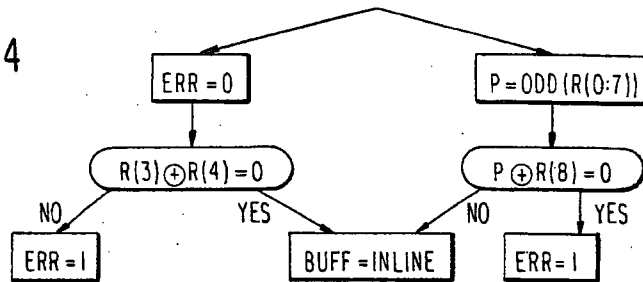


FIG. 5a
ELIMINATE COMMON
TERMS

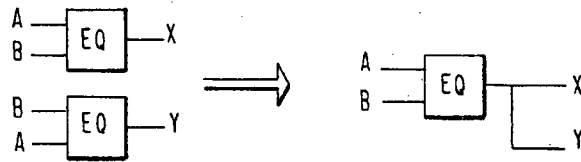


FIG. 5b
SIMPLIFY PARITY
OPERATORS

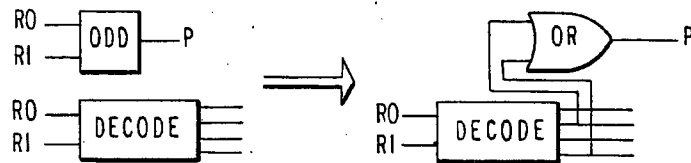
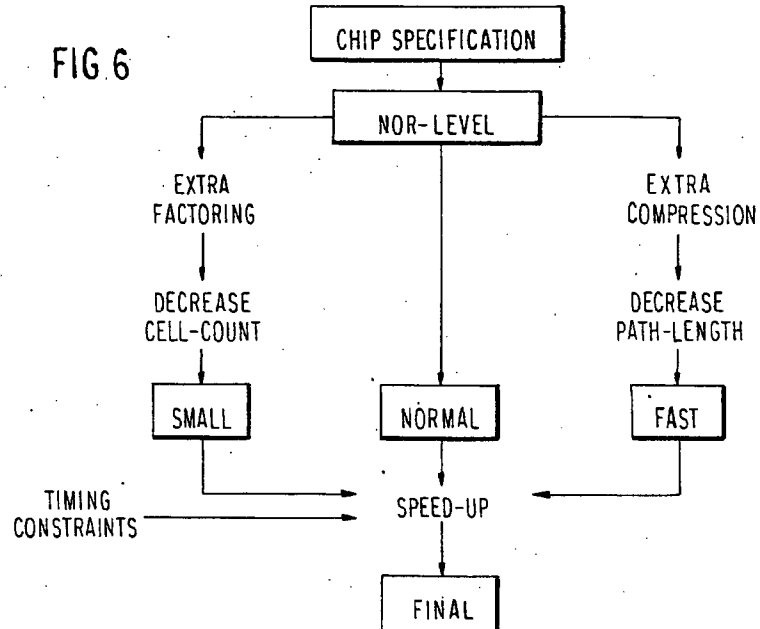
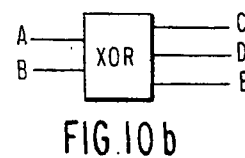
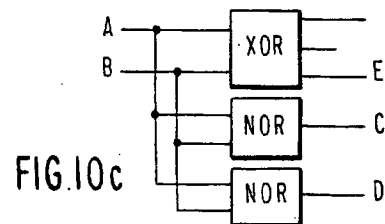
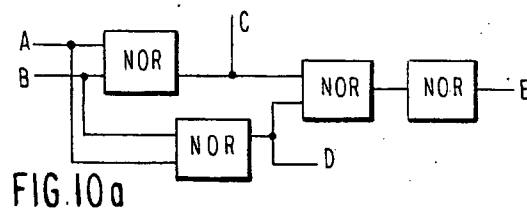
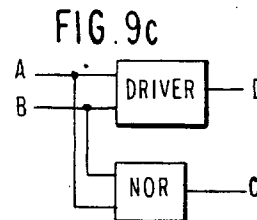
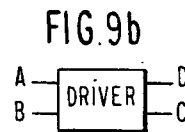
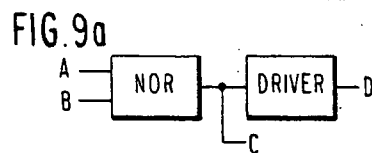
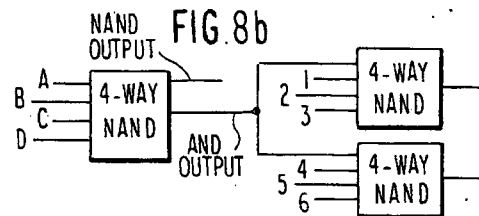
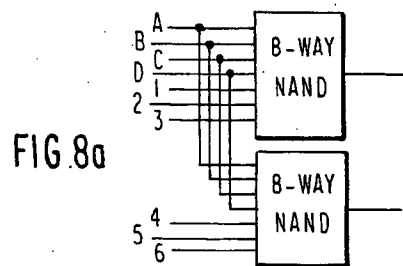
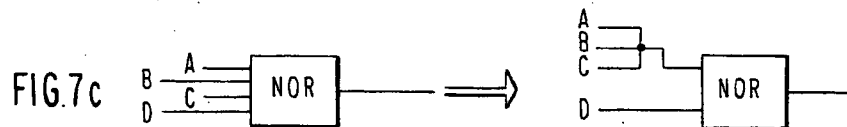
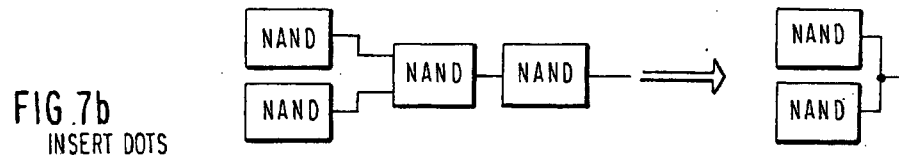
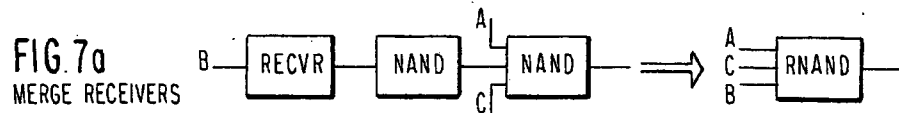


FIG. 6



U.S. Patent Oct. 27, 1987 Sheet 5 of 5 4,703,435



4,703,435

1

LOGIC SYNTHESIZER

BACKGROUND OF THE INVENTION

This invention is directed to logic design, and more particularly to a method of automated logic design.

As the complexity of processors has increased, the task of processor logic design has become more difficult. The designer may begin by designing a flow chart or other register-transfer level description to describe the intended operation of the processor, and the processor operation is then simulated from this description in order to ensure that a processor operating in accordance with the flow chart will provide the desired results. A logic implementation is then designed to achieve the operation described in the flow chart, and the resulting logic diagram and original flow chart specification are compared to ensure consistency. Finally, a physical layout is designed in accordance with the logic diagram implementation.

The above process has become significantly more difficult and extraordinarily time consuming with the increasing complexity of the processors being designed. For example, each chip in the 3081 processor available from International Business Machines Corporation includes over 700 circuits capable of performing extremely complex functions. The flow chart specification of such a processor will be quite complex, and even a first attempt at a logic diagram implementation will require a substantial amount of time. Further, with increasing processor complexity, the competing interests of gate count and timing constraints become increasingly difficult to satisfy. More particularly, a typical timing constraint may be that a signal must be provided from the output of register A to the input of register B within some predetermined period of time, and the designer may first propose a logic arrangement intended to satisfy this timing constraint while using a minimal number of gates in the circuit path between registers A and B. After timing analysis, however, it may be discovered that the timing constraint has not been satisfied, and the designer must then revise the arrangement of logic between the registers A and B, e.g., by using a larger number of gates to improve the processing speed in that area. Several iterations of design may be required before a logic design is obtained which indeed satisfies all timing constraints with the minimum gate count, and it is therefore not uncommon for the logic design to be quite costly in terms of engineering time.

In view of the above, there has been significant recent activity in the field of automatic logic synthesis. Early work centered on developing algorithms for translating a boolean function into a minimum 2-level network of boolean primitives, and extensions were developed for handling limited circuit fan-in and alternative cost functions. However, because these algorithms employ 2-level minimization, the time required to implement these algorithms increases exponentially with the number of circuits. The use of such algorithms therefore becomes impractical in designing large processors.

Other efforts have attempted to raise the level of specification, e.g., by beginning with behavioral specifications and producing technology-independent implementations at the level of boolean equations. However, the results of such techniques were usually more expensive than manual implementations and did not take advantage of the target technology. For example, the

2

system described by T.D. Friedman et al, in "METHODS USED IN AN AUTOMATIC LOGIC DESIGN GENERATOR (ALERT)," IEEE Trans. on Computers, Vol. C-18, No. 7 pp. 593-614 (1969), produced an implementation for an IBM 1800 processor which required 160% more gates than the manual design for that processor. Several attempts have been made to produce more efficient logic and to give the designer more control over the implementation, e.g., as described by: H. Schorr, "Toward the Automatic Analysis and Synthesis of Digital Systems," Ph.D. Thesis, Princeton University, Princeton, NJ, 1962; C.K. Mestenyi, "Computer Design Language Simulation and Boolean Translation," Technical Report 68-72, Computer Science Department, University of Maryland, College Park, MD, 1968; F.J. Hill and G. R. Peterson, *Digital Systems: Hardware Organization and Control*, John Wiley & Sons, Inc., New York, 1973. However, this control has resulted in specification language constraints, so that the specification is at a fairly low level and in closer correspondence with the implementation. This necessarily decreases the advantage of an automated approach, bringing it closer to a system for logic entry rather than logic synthesis.

Several tools have been developed to support the early part of the design cycle, e.g., as described in: M. Barbacci, "Automated Exploration of the Design Space for Register Transfer Systems," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1973; D. E. Thomas, "The Design and Analysis of an Automated Design Style Selector," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1977; E. A. Snow, "Automation of Module Set Independent Register-Transfer Level Design," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1978; L. J. Hafer and A. C. Parker, "Register-Transfer Level Digital Design Automation: The Allocation Process," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 213-219; A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System - An Example of Automated Data Path Design," *Proceedings of the Sixteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 73-80. The technique described in the last-cited publication began with a functional description of a machine and produced an implementation in two technologies of the registers, register operators and their interconnections, but not the control logic to sequence the register transfers. For both TTL and CMOS implementations, however, the automated implementation required substantially more chip area than existing manual designs.

There has also been recent work in logic remapping, i.e., transforming existing implementations from one technology to another. S. Nakamura et al S. Nakamura, S. Murai, C. Tanaka, M. Terai, H. Fujiwara, and K. Kinoshita, "LORES-Logic Reorganization System," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 250-260; describe a system which will help a designer translate an existing small-or medium-scale integration implementation into large-scale integration. However, remapping usually involves one-to-one substitution of new technology primitives for old technology primitives, and this often fails to take advantage of simplification which may be available at a higher technology-independent level.

3

4,703,435

4

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an automated logic synthesis technique which overcomes the above-described drawbacks. It is a more particular object of the present invention to provide such an automated logic synthesis technique which is capable of operating at a relatively high speed while achieving end results comparable to those obtained by manual design. It is a still further object of this invention to provide such an automatic logic synthesis technique capable of achieving satisfactory results in a number of different technologies.

Briefly, these and other objects of the invention are achieved by a logic synthesis method in which a register-transfer level flowchart specification is translated in a straightforward manner into a simple AND/OR logic implementation. After expanding the logic implementation to elementary representation and then applying textbook simplifications, the simplified AND/OR implementation is translated to a NAND or NOR implementation, depending on the target technology. The NAND or NOR implementation is then simplified by applying a sequence of simplification transformations which have been found by the present inventors to achieve satisfactory results, with the transformation sequence being modified to achieve "normal," "fast" or "small" logic designs. After simplification at the NAND/NOR level, the logic implementation is then translated to the target technology and further simplified. The result is an interconnection of the primitives of the target technology in a language from which automated logic diagrams can be produced in a known manner, and which can be submitted to existing programs for automated placement and wiring and chip fabrication.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more clearly understood from the following description in conjunction with the accompanying drawings, wherein:

FIG. 1 is a conceptual diagram of the logic synthesis technique according to the present invention;

FIG. 2 is a chart illustrating the multiple levels of simplification in the logic synthesis technique according to the present invention;

FIGS. 3(a)-3(p) illustrate simplifying transformations at the NAND/NOR level;

FIG. 4 is a simple illustration of a portion of a flowchart specification from which the present invention begins;

FIGS. 5(a)-(b) illustrate simplifications which may be performed at the AND/OR level;

FIG. 6 is a diagram illustrating the different scenarios of simplification at the NAND/NOR level;

FIG. 7(a)-7(c) illustrate examples of simplification at the hardware level; and

FIGS. 8(a)-8(b), 9(a)-(c) and 10(a)-10(c) illustrate further examples of technology-specific hardware simplifications.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The logic synthesis method according to the present invention is generally illustrated in FIG. 1. Previous publications describing some aspects of the system according to this invention, all of which are incorporated herein by reference, are: J. A. Darringer and W. H.

Joyner, "A New Look at Logic Synthesis," *Proceedings of the Seventeenth Design Automation Conference* Minneapolis, MN, 1980, pp. 543-549; J. A. Darringer, W. H. Joyner, L. Berman, and L. Trevillyan, "Experiments in Logic Synthesis," *Proceedings of the IEEE International Conference on Circuits and Computers ICC80*, Port Chester, NY, 1980, pp. 234-237A; J. A. Darringer, W. H. Joyner, C. L. Berman and L. Trevillyan, "Logic Synthesis Through Local Transformations," *IBM Journal of Research and Development*, Vol. 25 No. 4, July 1981.

The present invention is an automatic replacement for part of the manual design process. It operates on a logic design at three levels of abstraction. It begins with an initial implementation generated in a straightforward manner from the specification. The implementation can be simplified at this level, and then moved to the next level. This simplification is accomplished by transformations, either locally or globally, to achieve the simplification or refinement. By being able to operate on the implementation at several levels, the system can often make a small change at one level that will cause a larger simplification at a lower level. By using function-preserving transformations, it is ensured that in all cases the implementation produced will be functionally equivalent to the specified behavior. The inputs to the system illustrated in FIG. 1 are a description, in a register-transfer level, flow chart-control language, of logic functions to be implemented on a chip in a specified master slice technology, together with the interface constraints and a technology file which characterizes the target technology. The output of the system is a detailed interconnection of the primitives of the target technology in a language from which automated logic diagrams (ALD's) may be produced and which can be submitted to existing programs for automated placement and wiring and chip fabrication. The output implementation is in terms of the target technology and satisfies technology-specific constraints. Some timing or other physical problems may not be detectable before placement and wiring, and in such cases the synthesis process is repeated with a revised specification or modified constraints until an acceptable implementation is achieved.

The method according to this invention comprises PL/I programs operating on a representation of the logic in a data management system. The data management system is preferably that described by F. E. Allen et al, "THE EXPERIMENTAL COMPILING SYSTEM," *IBM Journal of Research and Development*, Volume 24 (1980), pages 695-715. The logic synthesis data base uses a single organization component referred to as a "box," with each box having input and output terminals which are connected by wires to other boxes. Each box also is designated by a type, which may be a primitive or may reference a definition in terms of other boxes. Thus, a hierarchy of boxes can be used, and an instance of a high-level box such as a parity box can be treated as a single box or expanded into its next-level implementation when that is desirable.

The logic synthesis data base is made of two groups of tables. The first group describes the technology being used, and is created from a technology file containing, for each box type, information such as name, function and number and names of input and output pins. These data are created in batch mode and read during initialization of the interactive system.

4,703,435

5

The second group of tables contains the representation of the logic created by the system. This group consists of a box table, a signal table and a set of auxiliary tables which describe the relationship between the boxes and the signals. There is some intentional redundancy in the data, i.e., each box has a complete list of input and output signals and each signal has a source and a list of sinks. Every box table entry contains type information which provides a link to the technology group, thus allowing programs to obtain technology information about a specific box.

Using the system generally illustrated in FIG. 1, a synthesis process according to the present invention may follow the sequence of steps shown in FIG. 2. FIG. 2 illustrates the three essential levels of description used in the method of the present invention: the initial AND/OR level 104, a NAND or NOR level 106 (depending on the target technology), and a hardware level 108 in which the types of the boxes are books or primitives of the target technology. At every level, the implementation is a network of boxes connected by signals. The purpose of this type of implementation is to find a set of transformations and a sequence of applying these transformations such that the original functional specification could be transformed by a sequence of small steps into an acceptable implementation.

As pointed out above, the process of this invention begins at step 100 with a register-transfer level description e.g. of the type shown in FIG. 4. The description consists of two parts: a specification of the inputs, outputs and latches of the chip to be synthesized; and a flowchart-like specification of control, describing for a single clock cycle of the machine how the chip outputs and latches are set according to the values of the chip inputs and previous values of the latches. At step 102 in FIG. 2, the register-transfer level description undergoes a simple translation to an initial implementation of AND/OR logic. This AND/OR level is produced by merely replacing specification language constructs with their equivalent AND/OR implementations in a well known manner, e.g., as described in J. R. Duley, "DDL - A Digital Design Language," Ph.D. Thesis, University of Wisconsin, Madison, WI, 1968; or J. A. Darringer, "The Description, Simulations and Automatic Implementation of Digital Computer Processors," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1969. At this first level 104 in FIG. 2, the logic begins in the form of an interconnection of boxes designated by types representing the operations which the perform, e.g., AND, OR, NOT, PARITY, EQ, XOR, DECODE, REGISTER (generic latch), SENDER, RCVR. At step 104 in FIG. 2, the initial AND/OR implementation is first expanded by taking all operators more complex than AND, OR or NOT and replacing these more complex operators with combinations of AND, OR, and NOT. Beginning with this expanded AND/OR logic, simplification is achieved by invoking PL/I program transformations which search for patterns of interconnected primitives and replace them by functionally equivalent patterns which are simpler in that they use fewer instances of operators, fewer connections, etc. The transformations at the AND/OR level 104 are local, textbook simplifications of boolean expressions, most of these simplifications reducing the number of boxes but not producing a normal form. Examples of simplifications are shown in FIGS. 5(a) and 5(b). Some of these transformations are similar to optimizing compiler techniques, e.g., constant propaga-

6

tion (moving "0" or "1" signals through logic blocks), common term elimination (combining blocks which compute the same function), combining nested associativecommutative operators, eliminating single input AND's and OR's, etc. Further examples of transformations used are as follows:

NOT(NOT(a))→a
AND(a, NOT(a))→0
OR(a, NOT(a))→1
OR(a, AND(NOT(a),b))→OR(a,b)
XOR(PARITY(a₁, . . . , a_n), b)→
PARITY(a₁, . . . , a_n, b)
AND(a, 1)→a
OR(a, 1)→1

These transformations may leave fragments of logic disconnected, and this can be cleaned up in a manner similar to the way in which compilers perform dead-code elimination.

After simplification at the AND/OR level 104, the simplified AND/OR implementation is transformed into a NAND or NOR implementation. Whereas AND/OR logic requires the use of multiple different operators in a logic design, NAND or NOR logic requires fewer operators, i.e., in a NAND logic design all logical functions can be implemented using some combination of only NAND gates. Whether a NAND or NOR implementation is produced is dependent upon the primitives available in the target technology. However, the NAND or NOR description is not technology-specific, in that there are no fan-in or fan-out restrictions. (Fan-in refers to the number of signals coming into a box, and fan-out refers to the number of sinks or destinations of a signal.) The transition to these primitives is accomplished naively by local transformations and may introduce unnecessary double NANDs or NORs which will be eliminated later. Also at this point, the chip interface information is used to place generic, i.e., not technology-specific, senders and receivers on the chip inputs and primary outputs, and to insert inverters where necessary to ensure the correct signal polarities. Techniques for accomplishing this transformation are well-known and need not be described here in detail.

At step 106 in FIG. 2, simplifying transformations are applied to each signal in the network. The NAND and NOR transforms are more difficult, and extensive experiments by the present inventors at the NAND/NOR level have resulted in a sequence or "scenario" of transformations which will produce acceptable results. The transformations are local in that they replace a small subgraph of the network (usually five or fewer boxes) with another subgraph which is functionally equivalent but simpler according to some measure. These transformations attempt to reduce the number of boxes of the implementation without increasing the number of connections. To accomplish this, the transformations must check the fan-out of the various signals involved, since this will affect the number of boxes and signals actually removed. Some of the transformations attempt to remove reconvergent fan-out which contributes to untestable stuck faults.

Some of the transformations are applied throughout the network in a number of iterations, preferably until no more transformations apply. FIGS. 3(a)-3(n) illustrate the NAND transformations NTR1 thru NTR10 used in one embodiment of this invention, and the NOR transformations would be identical except for the operator. Each transformation has an associated condition that determines if the replacement will simplify the

4,703,435

7

implementation by reducing boxes or connections. These conditions depend on the fan-out of the intermediate signals and on whether the target technology is assumed to have dual-rail output.

Experiments with the NAND/NOR level transformations have resulted in a normal sequence or "scenario," of transformations which have produced acceptable results. A "fast" scenario was developed which resulted in shorter path lengths, and a "small" scenario was also developed to obtain smaller designs. These are generally indicated in FIG. 6. In the preferred embodiment of this invention, the sequence of steps in the normal NAND/NOR scenario would be as follows:

APPLY GENNOR; (or APPLY GENNAND);
UNTIL NOCHANGE APPLY NTR1, NTR2,
CLEANUP, NTR3, NTR4, NTR10, CLEANUP,
NTR7, NTR9, PROPCON, CLEANUP, CTE,
CLEANUP;

FANIN 4;

APPLY NTR6A, FACTORN, NTR6A, CLEANUP;
APPLY NTR10, CLEANUP, NTR7 (NOIN-
CREASE), NTR9, PROPCON, CLEANUP;

APPLY CTE, CLEANUP; FANIN 8;

APPLY NFANIN, NTR11, CLEANUP;

The GENNOR or GENNAND transformations merely transform the AND/OR implementation into either NAND or NOR logic in accordance with the target technology. This type of transformation is well understood in the art and need not be described in detail here.

NTR1 in FIG. 3(a) removes double inverters and always applies, since it is always considered desirable to reduce the number of cells, and because this transformation does not increase connects or path lengths. This transform, and others, may in some instances increase fan out, but the fan out can be reduced, if necessary, at a later point.

NTR2 in FIG. 3(b) applies only if s_1 has no fan out and s_2 fans out only to primitives, i.e., either NANDs or NORs. This transform will not apply if it will result in an increase in the number of connects. For example, in the transformation illustrated in FIG. 3(b), gates 10 and 12 are eliminated and their corresponding input and output connections are also eliminated. However, if s_2 fans out to four NANDs, it would be necessary to apply the NTR2 transformation to each one, resulting in an increase in the number of connects.

NTR3 in FIG. 3(c) applies only if none of the gate outputs s_i fans out, s_r does not fan out, and no gate B_i exceeds the fan-in threshold for a single-cell book. This helps set up later dotting.

NTR4 in FIG. 3(d) removes redundancy locally. Redundancy is a property of a combinational logic circuit, and is present when the network contains a signal that can be set to a constant value without changing the function of that network. NTR4 also replicates logic if the output s of gate 12 fans out.

NTR6A in FIG. 3(g) sets up dotting and is only run if dotting is allowed in the target technology.

NTR7 eliminates some forms of redundant connections. This transform will replicate boxes, if necessary, unless the parameter NOINCREASE is specified. NTR7 actually comprises three transforms illustrated in FIGS. 3(h)-3(j), all of which are run each time NTR7 is called for in the above program.

NTR9 in FIG. 3(i) handles cases where a signal and its negation both go to a NOR or NAND gate. The "0"

8

input to gate 14 will be a "1" for the equivalent NOR transformation. This transform should be followed by PROPCON, described below.

NTR10 includes two different transforms illustrated in FIGS. 3(m) and 3(n), both of which are run each time NTR10 is called for. The NTR10 transform is run only if the outputs of gates 18 and 20 and FIG. 3(n) do not fan out.

NTR11 in FIG. 3(o) makes all generic registers (considered to have the OR function) have a fan-in of 1 by preceding each register with an appropriate number of primitives.

PROPCON, CLEANUP and CTE are analogous to the compiler operations of constant propagation elimination, dead-code elimination and common sub-expression elimination, respectively. Common sub-expression elimination, or common term elimination, refers to locating boxes which produce the same logic value, eliminating one box, and sharing the output of the other box.

FANIN 4 does not in itself perform any transformation but instead sets a variable known as "FANIN" to a value of 4.

FACTORN examines only boxes exceeding the FANIN limitations specified by the variable FANIN. It then applies the transformation of FIG. 3(p). This transformation will not reduce all boxes to below the specified FANIN limit, but only those boxes to which it applies by finding common sinks.

NFANIN corrects the fanin to the specified limit by building fanin trees which it constructs to have the fewest boxes and then to lengthen as few paths as possible.

In a NOCHANGE loop, the transformations are repeatedly run in their specified order until no further change in the logic occurs. In general, the order of the transformations and their inclusion in the NOCHANGE loop is such that succeeding transformations are invoked when preceding transformations can cause them to apply. For example, in the first loop, the sequence beginning with NTR9 is used to remove gates having complementary inputs. Since this can produce constant zeros or ones, constant propagation (PROPCON), removal of unconnected boxes (CLEANUP), common term elimination (CTE), and then more CLEANUP (to deal with now-unconnected common terms) must be run. On the other hand, after fan-in correction by factoring and NFANIN, some transformations should not be run, because they may destroy the fan-in limits already enforced.

In looking again at the program above, it can be seen that certain sequences of functions are performed, with some functions comprising a plurality of transformations. More particularly, with regard to the first NOCHANGE loop, transformations NTR1, NTR2, CLEANUP, NTR3 operate to reduce logic depth, i.e., number of levels of logic from input to output, with NTR1 reducing logic depth from two levels to one and NTR2 reducing logic depth from three levels to one. NTR3 at first glance appears to provide no depth reduction, since it transforms three levels of logic to three levels of logic. However, in some instances the last level, gate 11, can be subsequently eliminated, so that NTR3 is often useful in reducing logic depth.

Reducing logic depth, i.e., compressing the logic into fewer levels, will increase the chance of detecting redundancy. Thus, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP applied to remove redundancy.

4,703,435

9

After removing redundancy, a common terms elimination sequence CTE, CLEANUP is run.

After the NOCHANGE loop has finished running, transformations are applied to introduce dot patterns and to reduce fan-in to a specific level. This is accomplished by the step FANIN 4 which sets the fan-in limit to a value of 4, followed by the sequence NTR6A, FACTORN, NTR6A, CLEANUP, which serves to reduce fan-in at the expense of logic depth.

Once again, the introduction of dot patterns and the factoring to reduce fan-in may result in redundancy, so that the redundancy removal sequence NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP is applied.

Common terms are then eliminated by running CTE, CLEANUP.

Finally, the logic must be adjusted to the maximum fan-in value permitted by the target technology, e.g., a fan-in value of 8. This is achieved by applying FANIN 8 to set the fanin value at 8 followed by NFANIN, CLEANUP.

As should now be appreciated, the above program can be functionally represented as follows:

- A. LOGIC DEPTH REDUCTION LOOP
 - A1. REDUCE LOGIC DEPTH
 - A2. REMOVE REDUNDANCY
 - A3. ELIMINATE COMMON TERMS
- B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
- C. REMOVE REDUNDANCY
- D. ELIMINATE COMMON TERMS
- E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

The operations subsequent to the logic depth reduction loop may tend to expand the logic depth, so that the above process can generally be seen as a compression stage followed by an expansion stage. While it may be theoretically possible to obtain maximum logic depth reduction through two-level boolean minimization, this would compress the logic so far that re-expansion to take advantage of other simplifying transformations, e.g., at the subsequent hardware simplification, would be much more difficult. Thus, the logic compression transforms have been found particularly suitable.

The program set forth above concerns a normal scenario, and the "fast" and "small" scenarios can be obtained by modifying the above program as follows: for the small scenario, the following additional NOCHANGE loop is inserted after the NOCHANGE loop in the normal scenario:

UNTIL NOCHANGE APPLY NTR6, NTR5, NTR1, NTR2, CLEANUP, NTR3, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON CLEANUP, CTE, CLEANUP;

NTR5 in FIG. 3(e) applies only if the number of cells does not increase, and NTR6 in FIG. 3(f) applies only if the number of cells is decreased. Inspection of NTR5 and NTR6 shows that they can increase path length, and they are consequently only used in the small scenario. The other transformations in the added loop are provided to act on any changes which may result from NTR5 and NTR6. For example, NTR5 and NTR6 can produce double inverters, so the sequence beginning with NTR1 is run. NTR1 eliminates double inverters, and can introduce situations where other transforms apply.

10

Examination of the second NOCHANGE loop set forth above will reveal that the loop includes a first sequence NTR6, NTR5 for reducing the cell count by increasing the logic depth. The sequence NTR1, NTR2, CLEANUP, NTR3 is then applied to mitigate the logic depth reduction by taking advantage of transforms made available by NTR6, NTR5. After this logic depth reduction sequence, the redundancy removal and common term elimination sequence are applied in the first NOCHANGE loop.

Thus, the program for the "small" scenario can be written:

- A. LOGIC DEPTH REDUCTION LOOP
 - A1. REDUCE LOGIC DEPTH
 - A2. REMOVE REDUNDANCY
 - A3. ELIMINATE COMMON TERMS
- A'. CELL COUNT REDUCTION LOOP
 - A1'. REDUCE CELL COUNT
 - A2'. REDUCE LOGIC DEPTH
 - A3'. REMOVE REDUNDANCY
 - A4'. ELIMINATE COMMON TERMS
- B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
- C. REMOVE REDUNDANCY
- D. ELIMINATE COMMON TERMS
- E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

While the "small" scenario is designed to emphasize minimization of gate count, the "fast" scenario is designed to emphasize shorter path lengths, sometimes at the expense of gate count. Path length refers to the delay along a path from a signal's source to one of its destinations. Usually, path lengths are measured from registers or primary chip inputs to registers or primary chip outputs. The result can be the number of boxes in the path or the estimated delay of that path in nanoseconds.

The fast scenario inserts a call to NTR8 as the last step run in the first NOCHANGE loop. Immediately thereafter, FANIN is set to a value of 8 rather than 4, and NTR11 is omitted from the last line of the program. The significance of these changes to the program is as follows:

NTR8 in FIG. 3(k) is used in the fast scenario because it shortens paths. This may sometimes be at the expense of cells, however, since some of the boxes shown in FIG. 3(k) may have to be replicated. The factoring to a fanin of 8, also produces shorter paths, but may increase the cell count, e.g., in a dual-rail technology in which a 4-way NOR/OR required one cell and an 8-way required two cells. This will be explained in more detail with reference to FIGS. 8(a) and 8(b).

In a particular technology, there may be a number of different primitives or "books" having different fan-in capabilities, and different books may include different numbers of cells. For example, an 8-way NAND gate may use two cells while a 4-way NAND gate may use one cell. If 8-way NAND gates are used, e.g., to combine ten different inputs in two combinations with four inputs common to each combination, the result may be as shown in FIG. 8(a). Each book would receive seven inputs, and a total of four cells would be used.

If fan-in is limited to a value of 4, the same logic could be implemented as shown in FIG. 8(b) using three 4-way books. Although the number of books has increased, each book includes only one cell, so that the cell count decreases from four to three. However, the

4,703,435

11

cell count decrease is at the expense of increasing the logic depth by one level.

In the "normal" or "small" scenarios, it is worthwhile to set the fan-in value to 4 and to factor in an attempt to take advantage of the cell reduction which may be realized by using the smaller books. In the "fast" scenario, however, the increase in logic depth accompanying the use of the smaller books is unacceptable, and the fanin is instead set to the maximum allowable fan-in after the NOCHANGE loop.

Thus, the simplification program for the "fast" scenario can be functionally described as follows:

LOGIC DEPTH REDUCTION LOOP

- A1. REDUCE LOGIC DEPTH
- A2. REMOVE REDUNDANCY
- A3. ELIMINATE COMMON TERMS
- A4. REDUCE LOGIC DEPTH WHILE INCREASING CELL COUNT
- B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO MAXIMUM ALLOWED BY TECHNOLOGY
- C. REMOVE REDUNDANCY
- D. ELIMINATE COMMON TERMS
- E. ADJUST LEVEL TO MAXIMUM PERMITTED FANIN

The search strategy for the above transformations is to search the interconnected boxes of the data-base in sequence, looking for a pattern to which the transform may apply. The search is done for each transform in an efficient way, e.g., NTR2 searches the entire logic design for a one-input inverter, since this is faster than examining each multi-way NAND or NOR to determine if an inverter precedes or follows it.

After the simplification sequence described above, transformations are applied to the logic to map the NAND or NOR implementation to the target technology, and enforce technology-specific restrictions. This is performed at level 108 in FIG. 2. The transformations applied at level 108 may be generally described as follows, although their exact implementation will depend on the target technology

Technology-specific transforms may preferably be applied in the following order: first, generic NAND/NOR gates are mapped to their counterparts in the target technology. If the fan-in of a gate is too high and there is no corresponding primitive in the technology, a tree of primitives must be built to produce the same logical function. REG's, the generic latches, are mapped to the technology-specific latches. In general, the technology-specific latches have a limited number of pins for data values. If more data values are gated into the latch than can be accommodated, extra "ports" must be connected to the latch in a manner prescribed for the technology. SENDER's and RECEIVER's are mapped to their technology-specific counter parts.

Second, if the target technology is dual-rail, dual-rail books are introduced. With both positive and negative phases available from each gate, all inverters (except those on chip inputs) are removed and their output signals connected to the opposite phase of the source of their input signals. Third, technology-specific "tricks" are introduced, e.g., special books, drivers, receivers, etc., which were not known at the time of the generic transformation. These implement certain functions, such as XOR, combinations of driver and logic functions, combinations of receiver and logic functions, combinations of latch and receiver, etc., using fewer cells

12

than the primitive NAND or NOR implementation. The pattern of technology-specific NAND's or NOR's is searched for and replaced by the appropriate block. In FIG. 7(a), three cells can be replaced with a single NAND in the target technology having a built-in receiver.

If dots, i.e., wired AND's or OR's, are allowed in the target technology, patterns implementing +AND or +OR functions are located. If the inputs of these patterns have no fan-out, the pattern is replaced by a dot, e.g., as shown in FIG. 7(b). Dots can also be introduced to reduce fan-in as shown in FIG. 7(c). After dots are introduced, more special books may be present and are searched for again.

Next, fan-out is adjusted to meet constraints. Fanout limits are specified for technology-specific box types and output pins of those boxes. Fan-out is brought within these limits by replicating the violating box and distributing some of its fan-out to the copy (parallel repowering) or driving some of the fan-out with a repowering +OR or +AND function in front of the violating box (serial repowering). Additional dual-rail books are added after fan-out adjustment, but not so as to violate fan-out constraints.

Next, clock signals are introduced as chip inputs and distributed to the latches of the chip according to technology-specific requirements for clock distribution. Depending on the technology, this requires clock balancing and introduction of special clock drivers.

Next, path lengths back from latch-to-latch and from chip output to chip input are analyzed. Long paths are first shortened by rearranging fan-in and fan-out repowering, introducing dots (even at the cost of cell count), "undoing" factoring transformations performed at a higher level and introducing high-power books. Short paths are then padded to meet minimum path length requirements.

The fan-out adjustment is then repeated to correct fan-out violations which may have resulted from the path length correction.

Finally, scan-in and scan-out pins are introduced and the latches are linked together in an LSSD scan ring. A chip-in-place test and/or inhibit signals are introduced for chip outputs where required. Since this may introduce fan-out violations, fan-out adjustment is repeated.

An example of a hardware conversion and simplification program for a NOR technology following the above-described sequence may be as follows:

```

APPLY GENHW, CLEANUP;
APPLY DUAL (NO LIMIT), CLEANUP;
APPLY OPTDRIVE (SMALL OR FAST),
CLEANUP,
OPTXOR (SMALL OR FAST), CLEANUP;
APPLY GENDOT;
APPLY OPTDRIVE (SMALL OR FAST),
CLEANUP, OPTXOR (SMALL OR FAST),
CLEANUP;
APPLY FANOUT, DUAL, CLEANUP;
APPLY CLOCK;
APPLY TIMING;
APPLY FANOUT, DUAL, CLEANUP
APPLY SCANP, FANOUT;

```

GENHW maps generic gates to hardware primitives. Since fan-in has been adjusted at the end of the NAND/NOR simplification, much of this step is merely one-to-one mapping.

DUAL removes necessary inverters in dual-rail technologies by absorbing the inverters into other gates

13

which already have positive and negative phases available. This transform will normally be applied so as to exceed the fan-out limit. However, with the NOLIMIT option this transform will always apply.

OPTDRIVE takes advantage of a technology-specific book available, i.e., a driver book with built-in NOR capability. As shown in FIG. 9(a), the logic design may at this point include a NOR gate with a branched output with one branch going to a driver. Since both functions can be served by a single book in the target technology, the arrangement of FIG. 9(b) can be substituted. However, while this may be desirable in "normal" and "small" scenarios, there is a sacrifice in speed. Thus, for a "fast" scenario, the transformation is to the arrangement shown in FIG. 9(c) which provides for "parallel" operation and therefore higher speed at the expense of cell count.

OPTXOR takes advantage of a further technology-specific book, i.e., the XOR book. This transformation searches for a pattern of NOR gates providing the XOR function, e.g., as shown in FIG. 10(a), and substitutes the XOR book as shown in FIG. 10(b). Again, however, the transformation to FIG. 10(c) is employed in the "fast" scenario.

GENDOT introduces dotting in such a manner as to both eliminate gates and reduce fan-in. E.g., the transformation shown in FIG. 7(b) will eliminate gates 15 and 16 while the transformation shown in FIG. 7(c) will not eliminate gate 17 but will reduce the fan-in to that gate. This may save cells by permitting the use of a smaller book in the target technology and by allowing other transforms to apply. Since GENDOT changes the logic, OPTDRIVE and OPTXOR are applied again to search for more special books which may now exist.

FANOUT is applied to reduce the fan-out to the allowed limit. Note that the first half of the above hardware level simplification program is run without regard to fan-out limitations, as even the DUAL transform is applied with its NOLIMIT option. The various transformations may have caused fan-out violations which should be corrected by applying FANOUT in the manner discussed above. DUAL is then applied again, but this time so as not to violate fan-out constraints.

CLOCK is applied to distribute clock signals according to technology specific requirements in a manner known in the art.

TIMING is applied to correct path lengths by rearranging fan-in and fan-out trees, introducing more dots and changing power levels to shorten long path lengths, and inserting pad logic to lengthen the short paths, as necessary. After TIMING is applied, fan-out adjustment is again performed to correct any violations which may have resulted from the timing correction, and DUAL is again run, within fan-out constraints, to take advantage of changes made during fan-out adjustment.

Finally, SCANP is applied to link the registers in a LSSD scan path. Fan-out is again adjusted to correct any violations which may have resulted from SCANP.

The logic synthesis system of this invention employs three different levels of simplification between the original specification and the final implementation: high level simplifications, NAND/NOR level simplifications and technology specific simplifications. Several of the transforms at the three different levels are analogous, differing only in the types of boxes to which they apply, so that simplifications not made at one level would be caught later. This may appear redundant, but the application of transforms as early as possible reduces the size

4,703,435

14

of the implementation and helps prevent a greater explosion in size when, e.g., conversion to NANDs takes place.

A significant advantage of the present invention resides in its adaptability to more than one technology, requiring modifications to only a part of the system and leaving the technology-independent portions intact. This makes the synthesis process according to the present invention useful in synthesizing logic in a number of different technologies, and in fact facilitates the remapping from one technology to another in an efficient manner. Rather than merely mapping hardware primitives one-to-one from one technology to another, a first technology implementation is abstracted to a technology-independent level, e.g., from a TTL chip implementation to a NAND level implementation with generic registers, drivers and receivers. The NAND implementation can be mapped to a NOR level implementation in a straight-forward manner, with the NOR level simplification being performed in the manner described above with reference to level 106 in FIG. 2. The hardware mapping and simplification can then be performed in the manner described with reference to level 108 in FIG. 2. This enables the remapping to take advantage of simplifications which may be available at the NOR level.

Some of the work described in the earlier-cited publications concerned a synthesis process beginning with a behavioral description and producing technology-independent implementations of boolean equations. These processes did not take advantage of the target technology. Other work has centered on the synthesis of the data-flow portion of a machine, synthesis from a high-level behavioral description to a register-transfer description, and implementation of control logic in microcode or programmable logic arrays. In contrast, the present invention provides the following significant features:

First, the present invention uses local transformations at several levels of description, passing through technology-independent levels of description to a technology-specific description. This enhances the simplification while also facilitating the re-implementation of a design in a different technology.

Second, the specific sequences of simplifying transformations and the conditions associated with them have been found to provide acceptable results in normal, fast and small scenarios, thus making automated logic synthesis practical.

Further, timing, driver and other interface constraints are used at the hardware level to generate logic meeting these requirements.

Still further, the automated logic synthesis process according to the present invention greatly facilitates timing analysis and correction of the design to remove path length problems.

What is claimed is:

1. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;
simplifying said first logic design via said computer;

4,703,435

15

converting, via said computer, said simplified first logic design to a second logic design in a second logic system requiring fewer different logic operators than said first logic system, said second logic system comprising a plurality of interconnected cells and performing equivalent functions;

simplifying, via said computer, said second logic design, said step of simplifying said second logic design comprising the steps of: applying a depth reduction sequence of logic transformations for reducing the depth of said second logic design; and subsequently applying a size reduction sequence of logic transformations for reducing the size while possibly increasing the depth of said second logic design; and

converting, via said computer, said simplified second logic design to said desired technology.

2. A method as defined in claim 1, wherein said step of applying said depth reduction sequence of logic transformations comprises: applying a first logic transformation set for reducing logic depth; applying a second logic transformation set for reducing redundancy; and applying a third logic transformation set for eliminating common terms.

3. A method as defined in claim 2, wherein said step of applying said size reduction sequence of logic transformations comprises: applying a fourth logic transformation set for reducing logic size while possibly increasing logic depth; applying a fifth logic transformation set for reducing redundancy; and applying a sixth logic transformation set for eliminating common terms.

4. A method as defined in claim 3, wherein said second and fifth logic transformation sets include at least one common logic transformation. said common logic transformation being applied in said depth reduction sequence regardless of whether said common logic transformation will increase the number of cells in said second logic design and being applied in said size reduction sequence only if it will not result in an increase in said number of cells.

5. A method as defined in claim 2, wherein said step of applying said depth reduction sequence of logic transformations further comprises applying a fourth logic transformation set (e.g., NTR8), following said third logic transformation set, for further reducing said logic depth while increasing the number of cells in said second logic design.

6. A method as defined in claim 3, wherein said converting step comprises converting said simplified second logic design to a hardware logic design in said desired technology represented as a plurality of hardware primitives. said method further comprising the step of simplifying said hardware design, said hardware simplifying step comprising:

applying a first hardware transformation set for substituting technology-specific books for predetermined patterns of said hardware primitive; dotting signal lines to decrease the number of components in said hardware logic design, and to decrease fan-in in some portions of said hardware logic design even if the number of components in said portions is not decreased; applying said first hardware transformation set; correcting fan-out in said hardware logic design to a desired value; adjusting path lengths in said hardware logic design; and correcting fan-out to said desired value.

16

7. A method as defined in claim 1, wherein said step of applying said size reduction sequence of logic transformations comprises applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a first fan-in value.

8. A method as defined in claim 1, further comprising the step of applying a cell reduction sequence of logic transformations for reducing the number of cells in said second logic design, said cell reduction sequence being applied between said depth reduction and size reduction sequences.

9. A method as defined in claim 8, wherein said cell reduction sequence of logic transformations includes a first set of logic transformations followed by a second set of logic transformations, said second set of logic transformations comprising said depth reduction sequence of logic transformations.

10. A method as defined in claim 8, wherein said desired technology has a maximum allowable fan-in value, said step of applying said size reduction sequence of logic transformations comprising applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a desired fan-in value less than said maximum allowable fan-in value.

11. A method as defined in claim 10, further comprising the step of correcting the fan-in characteristics of said second logic design in accordance with said maximum allowable fan-in value subsequent to application of said size reduction sequence of logic transformations.

12. A method as defined in claim 1, wherein said depth reduction sequence of logic transformations is applied a plurality of times prior to applying said size reduction sequence of logic transformations.

13. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NAND logic.

14. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NOR logic.

15. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;

simplifying said first logic design via said computer; converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design, said second logic design comprising a plurality of interconnected cells and performing equivalent functions as said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware design comprising: applying a first hardware transformation

4,703,435

17

set for substituting technology specific components for predetermined patterns of said hardware; dotting signal lines via said computer, to decrease the number of components in said hardware logic design and to decrease fan-in in some portions of said hardware logic design even if the number of components in said portions is not decreased; adjusting path lengths in said hardware logic design via said computer; and correcting fan-out in said hardware logic design to a desired value via said computer.

16. A method as defined in claim 15, wherein said step of simplifying said hardware design further comprises applying said first hardware transformation set again after said dotting step but before said adjusting step.

17. A method as defined in claim 16, wherein said step of simplifying said hardware logic design further comprises the step of correcting said fan-out in said hardware logic design after said second application of said first hardware transformation set and before said adjusting step.

18. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail transformation for removing some of said inverters by substituting for said inverter and opposite-phase output signal available from its associated component, said dual-rail conversion transformation being applied both prior to said step of applying said first hardware transformation and subsequent to said step of correcting fan-out.

19. A method as defined in claim 18, wherein said dual-rail conversion transformation is applied prior to said step of applying said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, said dual-rail conversion transformation being applied after said step of correcting fan-out only to the extent that application of said dual-rail conversion transformation will not result in fan-out exceeding said desired value.

20. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail conversion transformation, prior to said step of applying said first hardware transformation, for removing some of said inverters by substituting for said some inverters an opposite-phase output available from their associated components.

21. A method as defined in claim 20, wherein said step of simplifying said hardware design further comprises the step of applying said dual-rail conversion transformation subsequent to said step of correcting fan-out.

22. A method as defined in claim 21, wherein said dual-rail conversion transformation is applied prior to said application of said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, and is applied subsequent to said step of correcting fan-out only to the extent that application of said dual-rail conversion transformation will not result in fan-out exceeding said desired value.

18

23. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating a first logic circuit design via said computer, in accordance with said description;

simplifying said first logic design via said computer; converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components and including inverters for receiving and inverting output signals from associated ones of said components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design comprising: applying a dual-rail conversion transformation for removing some of said inverters by substituting for said some inverters an opposite phase output signal available from their associated components, said dual-rail conversion transformation being applied without regard to the effect on fan-out characteristics of said hardware logic design; applying a first hardware transformation set for substituting technology-specific components; dotting signal lines in said hardware logic design; adjusting path lengths in said hardware logic design; correcting fan-out in said hardware logic design to a desired value; and applying said dual-rail conversion transformation only to the extent that it does not result in a fan-out exceeding said desired value.

24. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design accordance with said description;

simplifying said first logic design via said computer; converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design comprising selectively applying first or second hardware transformation sets for substituting technology-specific components for predetermined patterns of said hardware components, said first hardware transformation set resulting in fewer components than said second hardware transformation set and said second hardware transformation set resulting in higher-speed logic than said first hardware transformation set.

* * * * *

EXHIBIT

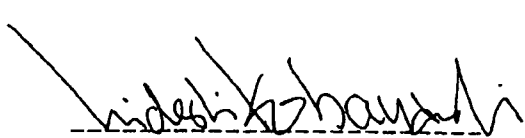
24

MANAGING VLSI DESIGN DATA
WITH
A RELATIONAL DATABASE SYSTEM

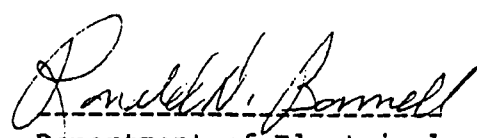
by

Yoon-Pin Simon Foo
Bachelor of Science in Electrical Engineering
University of South Carolina
Columbia, South Carolina, 1983

Submitted in Partial Fulfillment of the
Requirements for the Degree of Master of Science
in the College of Engineering
University of South Carolina
1984



Department of Electrical
and Computer Engineering
Director of Thesis



Department of Electrical
and Computer Engineering
Second Reader



Dean of Graduate School

CONFIDENTIAL

FOO 000038

To my Mom and Dad

For God so loves the world that He gave His only Son,
and whoever believes in Him shall not perish but have
everlasting life.

-- John 3:16

CONFIDENTIAL

FOO 000039

ABSTRACT

This thesis presents the use of a relational database system to manage very-large-scale-integration (VLSI) designs. The deficiencies of a commercial database system to handle CAD data is first discussed, and the need for a dedicated database management system (DBMS) is put forth. A multi-level abstraction of VLSI design is presented. At the design level, the notion of design hierarchy is introduced. At the representation level, the geometry (or layout) view and the interface-description view of design objects are developed. The collection of these individual views forms an enterprise schema. The enterprise schema is defined by an extended entity-relationship (EE-R) data model. The algorithm for conversions or mappings from the enterprise schema to a conceptual schema is presented. Normalization or decomposition of tables into their First Normal Forms (1NF) is introduced. The enforcement of integrity constraints on the design data is also discussed.

Using QUEL (QUERY Language) as a data manipulation language (DML), this research also investigates the complexity of query constructions. Simple and complex queries to retrieve design information using relational operators and aggregate functions are examined. The use of abstract data types (ADTs) to simplify query construction is also presented.

Next, the architecture of a novel integrated computer-aided-design (CAD) system comprising available design tools and a commercial relational DBMS, INGRES, is presented. Finally, a custom VLSI design example using a programmable logic architecture is presented, and the management of the chip design database is demonstrated. The novel implementation of a prototype programmable memory array (PMA) chip, PMA422, is also presented.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my adviser Dr. Hideaki Kobayashi for his untiring guidance and encouragement which made this thesis possible. My gratitude also goes to Professor Ronald D. Bonnell, Chairman of ECE Dept., and Dr. Michael N. Huhns, for their invaluable suggestions and comment.

I would like to thank my "adopted" family, the Riders, for providing all the love and inspirations during the preparation of this thesis. Last, but not least, I wish to express my deepest appreciation and gratitude to my parents, brothers and sisters in Malaysia for all their love and unfailing support throughout these years.

v

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	v
LIST OF FIGURES	viii
CHAPTER I INTRODUCTION	1
1.1 A Brief Overview of Logical Database Design.....	2
1.2 The Need for a DBMS for CAD Data.....	5
CHAPTER II MULTI-LEVEL DESIGN ABSTRACTION	10
2.1 Design Level.....	11
2.2 Representation Level.....	13
2.2.1 Extended E-R Data Model.....	14
2.3 View Modeling.....	19
2.3.1 Interface-Description View.....	20
2.3.2 Geometry View.....	25
CHAPTER III CONCEPTUAL SCHEMA	28
3.1 The Relational Data Model.....	28
3.2 Normalization.....	32
3.2.1 Integrity Constraints.....	36
CHAPTER IV RETRIEVING DESIGN INFORMATION	38
4.1 QUERy Language (QUEL).....	38
4.1.1 Relational Operators.....	38
4.1.2 Aggregate Functions.....	43
4.2 Applications of Abstract Data Types.....	44
CHAPTER V AN INTEGRATED CAD SYSTEM	56
5.1 A Relational DBMS:INGRES.....	56
5.2 A Collection of CAD Tools.....	58
5.2.1 Logic Synthesizer:SPAN.....	58
5.2.2 Channel Router:CHRP.....	59
5.2.3 Graphics Interface:GILL.....	59
5.2.4 Layout Editor:KIC.....	59
CHAPTER VI A VLSI DESIGN EXAMPLE	62
6.1 Programmable Logic for Parallel Convolution.....	62
6.2 Implementation of the Prototype PMA422 Chip.....	72
6.3 Managing the Chip Design Database.....	78

CHAPTER VII	CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH	84
7.1	Future Research Topics.....	83
REFERENCES		85
APPENDIX		87

LIST OF FIGURES

1. Logical Database Design Process.....	3
2. A Design Hierarchy.....	12
3. List of symbols used in the EE-R Data Model.....	15
4. Enterprise Schema for a Design Object.....	17
5. The Interface-Description View.....	21
6. The Geometry View.....	27
7. A 3-Transistor dynamic RAM cell.....	51
8. Logic schematic for a 4X4 CSA network.....	53
9. An Integrated CAD System.....	57
10. Box-level simulation of an adder network.....	61
11. Tree expression of subfunction for $Q = 4$	65
12. DNF architecture for $Q = 4$	66
13. (a) PMA for $n = 8$, $P = 4$, and $Q = 2$	68
(b) PMA for $n = 8$, $P = 8$, and $Q = 2$	68
14. CSA network for $n = 8$, $P = 4$, and $Q = 2$	69
15. (a) Plot of M versus P for $n = 8, 16, 32$	71
(b) Plot of N versus P for $n = 8, 16, 32$	71
(c) Plot of $(M + N)$ versus P for $n = 8, 16, 32$	71
16. Logic Schematic of PMA422.....	73
17. Program Table for PMA422.....	75
18. Floorplan of pmaBlk.....	76
19. Model for a Learning System.....	84

CHAPTER I

INTRODUCTION

Recently, there have been considerable interests in using commercial database systems for managing VLSI design data. Much emphasis is being placed on integrated CAD systems or workstations. However, most of the current CAD tools are not systems[1]. These collections of design tools do not have a common design database where the consistency of the design data is maintained. The heart of an integrated CAD system is a common database managed by a database management system (DBMS). The primary task of the DBMS in an integrated CAD system is to maintain the consistency between the different representations of VLSI designs, along with enforcing integrity constraints on the design data. However, due to the complex nature of design data, today's database systems are not well suited for VLSI applications.

1.1 A Brief Overview of Logical Database Design

A database is a computerized collection of operational data stored on disks or other storage media. A set of application programs are then written to manipulate the data with the following typical operations:

- o Retrieve (Look-up)
- o Update (Modify, Replace)
- o Insert (Append)
- o Delete

A schema is a plan of the database structure for a particular view of the data. An enterprise is a general term for any organization that exists and can be described.

An enterprise schema is the integration of all individual views into a single enterprise view. The entity-relationship (E-R) semantic data model is used to define the enterprise schema. The enterprise schema is then "mapped" by a specific data model to become a conceptual schema. INGRES, a commercial relational DBMS, uses a relational data model to define its conceptual schema.

The enterprise schema can be broken down into separate views by individual users for a particular need. Each user's view is then defined by a user schema, and represented by a data model. Figure 1 shows the structure of a typical logical database design process. Raw data about an enterprise is first collected and processed to fit

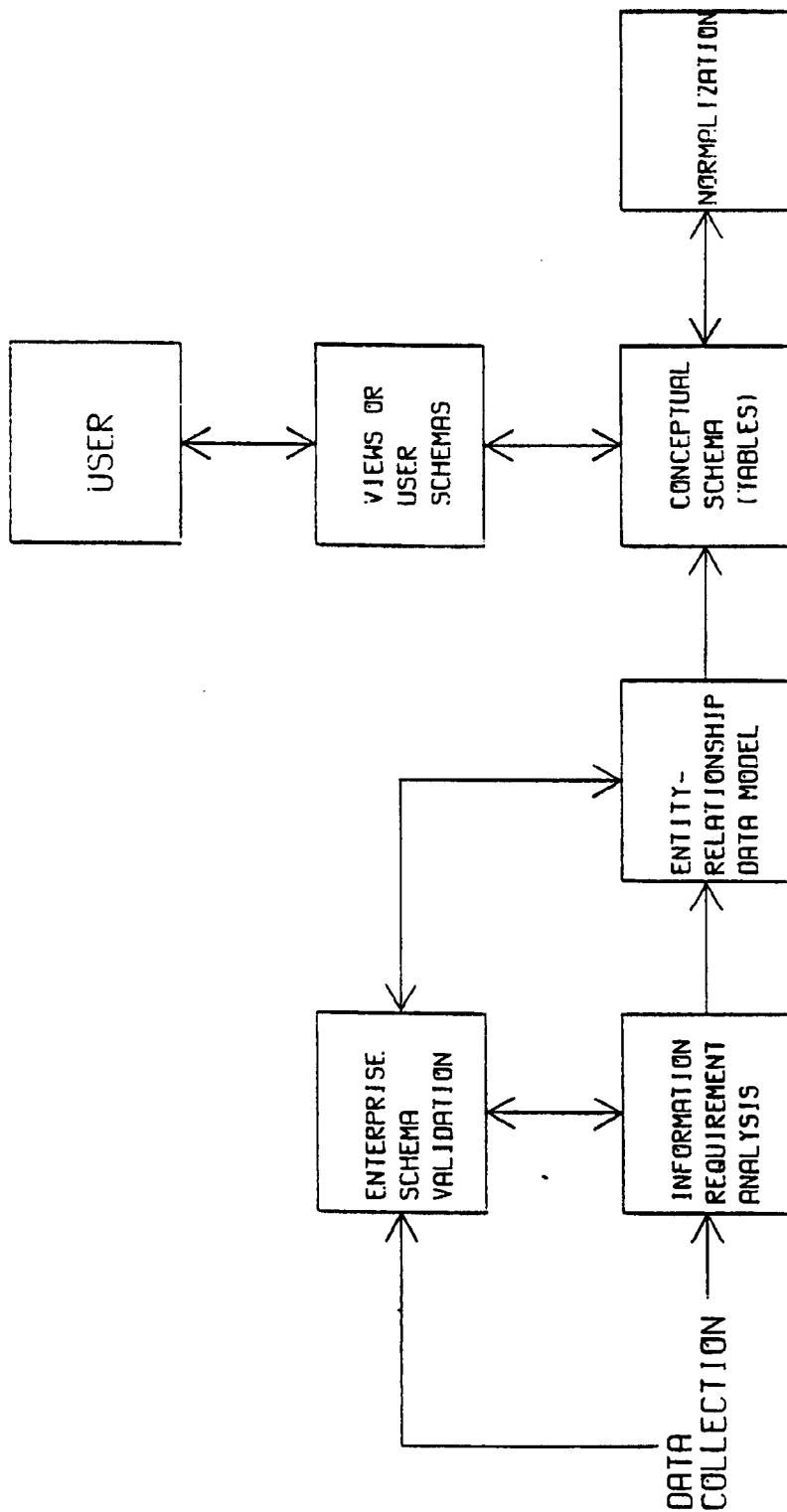


Figure 1. Logical Database Design Process

the criteria given by the information requirement analysis. An enterprise schema is built from the information collected. The enterprise schema is represented by the entity-relationship (E-R) data model. The conversions or mappings from the E-R model to a relational data model (RDM) result in a set of tables or relations. Tables with complex domains need to be normalized or decomposed into tables with simple domains. These simplified tables, called flat files, are then in their First Normal Form (1NF).

1.2 The Need for a CAD Database System

An integrated design database is needed before a collection of design tools can be formed into a CAD system. The database system organizes design information across different levels of representation, alternative implementations, and evolutionary versions. Particular data structures selected to represent a specific database formed a database schema. Each representation of design data is specified by a schema.

Design regularity reduces the design complexity as well as the size of the design database. An object is designed once and used frequently, and its description only appears once in the design database. Additional information may include how it is instantiated and placed within a design. VLSI circuits can be described in several representations, including functional descriptions, logic schematics, circuits, stick diagrams, and geometrical layouts. Each is appropriate for a particular phase of design. Geometries are used for mask-making and design rule-checking, while logic and transistor-level descriptions are used in simulation and timing verifications. Generally, VLSI designs are partitioned into architecture, logic, circuit, and layout. Each representation must be shown to be equivalent.

A design is a hierarchical process, and proceeds by a top-down decomposition of systems into subsystems, and a bottom-up integration of building blocks from more primitive building blocks. The design is completed when all subsystems can be implemented by existing building blocks and primitives. Design applications need to organize and structure the design data hierarchically, support multiple design representations, maintain design versions and alternatives, and enforce integrity constraints to ensure the consistency of the design. A chip design database contains information about how design objects are composed from primitives (such as geometries, transistors, and gates), how a design is represented (in layout, circuit, and logic), how the design has evolved over time, who is responsible for designing the parts, and so forth.

Since much of VLSI design is evolutionary or exploratory, the database must support design versions and alternatives. Versions are improvements or corrections to a design object, while alternatives are different implementations of the same object with varying performance characteristics. Versions, providing insights into design approaches and rationale, are needed for documentation purposes. Alternatives, especially within design libraries, enable designers to experiment with different implementations of the same function.

The object design data is made self-explanatory by placing interface descriptions, dependency information, and design responsibility. This helps a designer understand how an object is used, where it is used, and who is responsible for its design. Another mechanism, called the design validation subsystem, helps to keep the design consistent, identifying the portions of the design affected by changes. In other words, it ensures a complete propagation of a change to all parts of a design that depend on it.

A commercial DBMS can enforce only very simple integrity constraints. Due to the nature of most business applications, commercial DBMS's are designed to operate on simple domains. Therefore, constraints on data are well-defined and are relatively easy to enforce. Constraints on design data, however, are not easy to specify or enforce. For example, compare the constraint "salary must be greater than zero" with "the circuit must behave as specified with expected performance". This points out the complexity of enforcing integrity constraints on design data.

Commercial database systems are designed to handle large volumes of regularly-structured data. However, in VLSI applications, design data are often organized into complex structures with a large number of interconnected files of relatively small size. For example, in CAD applications, there is the need to support new data types

such as boxes, wires, polygons, and text strings where the storage structure is not regular. Hence, VLSI design data are not easily formatted for storage in a conventional DBMS. Obviously, there is a mismatch of what the design tools need and what the database system provides.

This research investigates the issues and processes of representing the design information in a common database with a commercial relational DBMS, INGRES. An integrated CAD system is developed, and an application example is demonstrated by the management of a prototype chip, PMA422.

Chapter II presents the multi-level abstraction of design data. At the design level, the notion of design hierarchy is introduced. At the representation level, the extended entity-relationship (EE-R) data model is used to describe the enterprise schema. Individual views of the design data, specifically the geometry and interface-description views are created.

Chapter III shows the conversions or mappings from the EE-R data model to a relational data model (or tables). A brief discussion on restrictions placed on the relations is presented. The enforcement of integrity constraints on design data is also demonstrated.

Chapter IV concerns the retrieval of design information. A brief introduction to INGRES' QUERy Language (QUEL) as a data manipulation language is presented. The use of relational operators and aggregate functions to build complex queries is studied. Applications of abstract data types to simplify query construction are also demonstrated.

Chapter V presents the architecture of a novel integrated CAD system, with a relational DBMS as the heart of the system. A brief insight into the characteristics of several existing CAD tools is also presented.

Chapter VI illustrates an application of the CAD system to a VLSI design example. The use of programmable logic for parallel convolution is presented as a background. The notion of function partition to realize an optimal memory array is discussed briefly. The novel implementation of a prototype chip called PMA422 is presented. The management of the chip design database is also presented.

Chapter VII summarizes the state of current research accomplishments, and provides an insight into some future work.

The Appendix contains a listing of the complete CIF design file of the prototype PMA422 chip.

CHAPTER II

MULTI-LEVEL DESIGN ABSTRACTION

Design information can be considered at two levels of abstraction, i.e., the design level and the representation level. A good reason behind this information partitioning is that some facts may be irrelevant to a user at a certain level of abstraction. For example, the designer at the functional level may not be interested in the geometrical data. So, the DBMS has to decide which relevant data should be transparent to the user at a certain design level. This is achieved by creating individual views of the whole enterprise schema. At the higher level (i.e. the design level) of abstraction, a design schema defines the design representations and the dependencies between them. Design objects are defined as occurrences of the design schema.

A design object can be described by its functional description, circuit diagram, and geometrical layout. All these representations describe the same object. However, some descriptions are derived from others in the design process. For example, the circuit representations are derived from the design specifications, and the layout is derived from its circuit representation. The consistency between the different descriptions must be maintained.

2.1 The Design Level

At the design level of abstraction, a VLSI design can be represented by functional specifications using text, by a circuit diagram, or by a geometrical layout. Each one of these descriptions can be called a representation. Representations which are independent of others are called primary representations, while others are the dependent representations. The dependencies between representations are directed and acyclic, thus forming a design hierarchy as shown in Figure 2. At the design level of abstraction, the dependencies between the representations are of primary concern. If the database is to contain meaningful data, then it is important to keep the representations consistent with respect to each other.

In the logical database, the design hierarchy is represented by a design schema. An occurrence of the design schema is called a design object. The design database is a collection of design schemas and design objects. The design schema is a description of a design object at the design level. At a lower level of abstraction, i.e., the representation level, the design object is described by a representation schema.

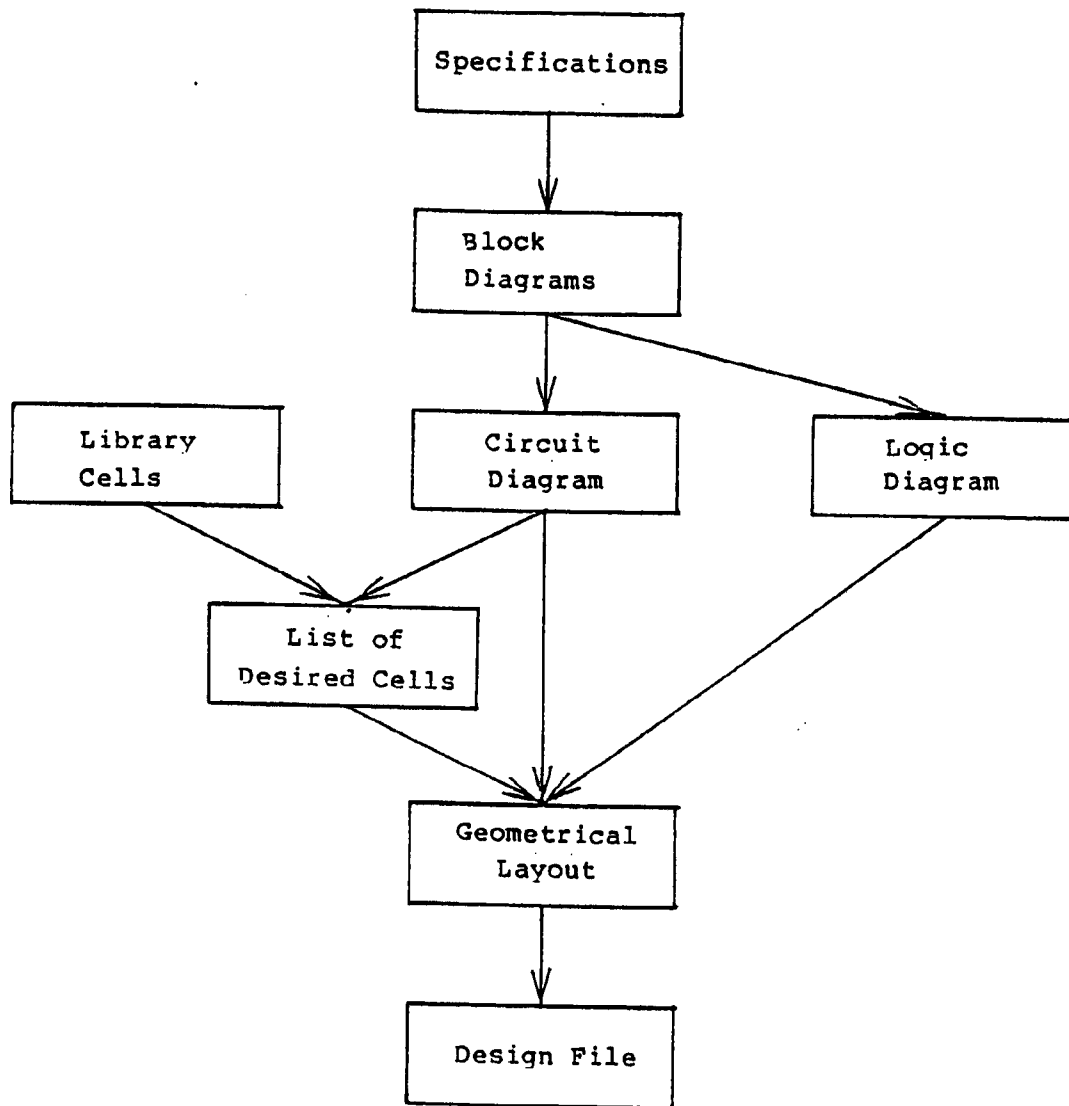


Figure 2. A Design Hierarchy

2.2 The Representation Level

Current data models were developed with commercial applications in mind [2]. Although it is possible to use this framework to model engineering objects, it is not suitable for expressing the complexity of VLSI designs explicitly. For example, consider the layout description of a shift register. The layout is composed of primitives such as "boxes", "wires", etc., of various processing layers, and so forth. Clearly, such a structure can not be conveniently described by a single entity.

In CAD applications, objects can be thought to be complex and composite. VLSI objects are often composed from other smaller and simpler objects. The term complex means that a large and variable number of entities are required to describe a single object, while composite means that an object can be composed of simpler but complex (and perhaps composite) objects.

Using the above framework, VLSI objects can be described by several entities and by their relationships between them. For the research reported herein, the EE-R data model is chosen to model the representation schema of a design object.

2.2.1 An Extended E-R Data Model (EE-R)

In the EE-R data model, design information can be modeled in terms of entities and relationships between the entities [2]. Entities are abstractions of objects and facts in general, and attributes are properties of an entity. Then a class of entities that can be described by the same attributes form an entity set. A key is a collection of one or more attributes that uniquely identifies an entity set. The relationship sets are represented by their attributes, and keys (if any). Relationships between the entities can be classified into two main categories, i.e., the regular and weak relationships. Figure 3 shows a list of symbols used in the EE-R data model.

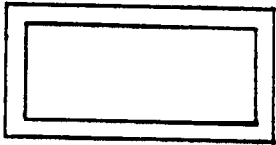
A weak relationship is a 1:N binary relationship, and the existence of the subordinate entities depends on the existence of a higher level entity. Thus these subordinate entities are referred to as owned entity class since they are owned by some higher level entities. However, to connect a simpler composite object to one or more higher level composite objects, another class of relationship called the reference class is needed. Simple objects such as "PlaCellPair" can exist in the database independent of any other objects. A composite object such as a "PLA" is composed of many simple (or even composite) objects. One entity set is chosen to represent composite objects.

- 14 -

Entity Classes



Kernel Class

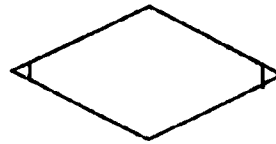


Owned Class

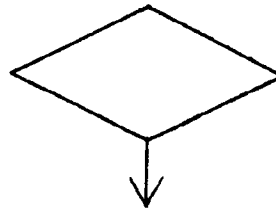


Regular Class

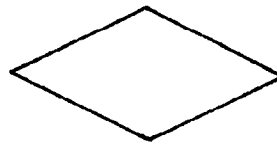
Relationship Classes



Reference Class



Weak Class



Regular Class

Figure 3. List of symbols used in the EE-R Data Model

The entity sets which identify composite objects form the kernel entity class. Thus, a composite object can be represented by an entity from the kernel class and all entities are connected to this entity by means of weak or reference relationships.

The reference relationship is constrained to be valid only between entities of the kernel class, since they express relationships between composite objects. Note the difference between the regular and reference relationship classes. Both are N:M relationships, but when two entities are connected by the regular relationship, it is not clear whether those two entities describe a composite object, or the two entities are just loosely connected.

Figure 4 shows the representation schema for an object at the representation level. Attributes are omitted at this point in order not to clutter the picture. Each object can be described by its function, geometry, and its interface description. In addition, each object is also described by its constituting objects and the connections between them.

Using the above EE-R data model, each object (simple or composite) is represented by an entity in an entity set of the kernel class. The object's function is an attribute of the object's entity. For example, if the function of the object "gate" is to "NAND" two input signals, then the attribute value of "function" is "2-input-NAND".

- 16 -

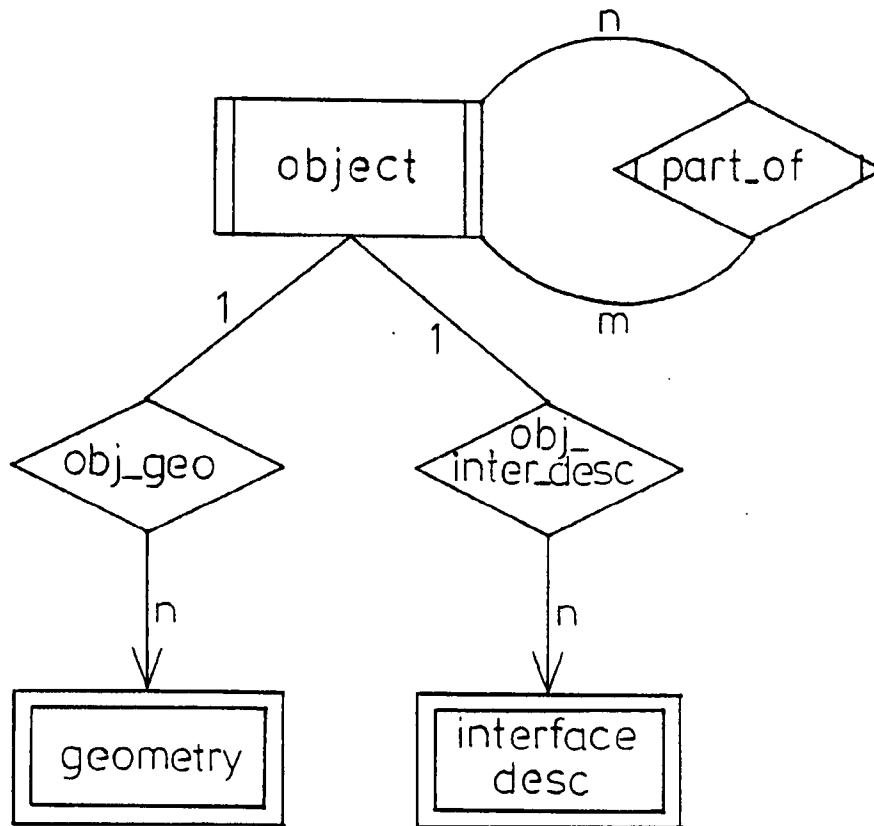


Figure 4. Enterprise Schema for a Design Object

The geometry and interface descriptions are described by entities of the owned class, which are then connected to the object's kernel class entity-set by weak relationships. The composition of an object is represented by the recursive reference relationships between the object's set of entities. The key attribute value (e.g. the object's id) of the object's kernel class entity-set uniquely identifies the complete object in a common design database.

2.3 View Modeling

The enterprise schema may be broken down into individual views of the design data. A good reason behind this information partition is that some facts about a particular design object may be irrelevant to a user at a certain level of abstraction. For example, the designer at the geometrical or layout level may not be interested in how the design object is interfaced with other objects at a higher level. So, the DBMS has to decide which relevant data should be transparent to a user at a certain design level. This is achieved by creating individual views of the enterprise schema. Each user's requirements are analyzed and represented with an EE-R data model. Associated with each view are entity sets, relationship sets, and attributes that describe the design objects being modeled. The individual user's requirements are then integrated into a single global view called the enterprise schema. Inconsistencies and redundancies among the different user views must be recognized and resolved. The result of view integration (the enterprise schema) becomes, after conversions, the conceptual schema. Two very important individual views developed in this research are the interface-description, and the geometry (or layout) views of the design object.

2.3.1 Interface-Description View

The interface-description of a VLSI cell tells the user how the cell can be connected to the "outside world", along with its constraints. At the representation level of design abstraction, the interface-description view aids in validating the connections of various design objects. Representation objects have interface descriptions to help keep them self-explanatory and consistent. They document the object and make connectivity information explicit so that an object can be used without a detailed understanding of its implementation.

Figure 5 shows an interface-description view of a design object based on the EE-R data model. The design object is represented by a kernel-class entity-set. The composition of the object is represented by the recursive N:M reference relationships. The variations in power, speed, and chip area are represented by a performance entity-set. Note that the binary "object-performance" relationship of 1:1 mapping class is a weak relationship. The "port" criteria is defined by another entity-set of the owned class. Similarly, the "object-port" binary relationship is a weak relationship, of 1:N mapping class. A brief description of each attribute in the interface-description view is as follows:

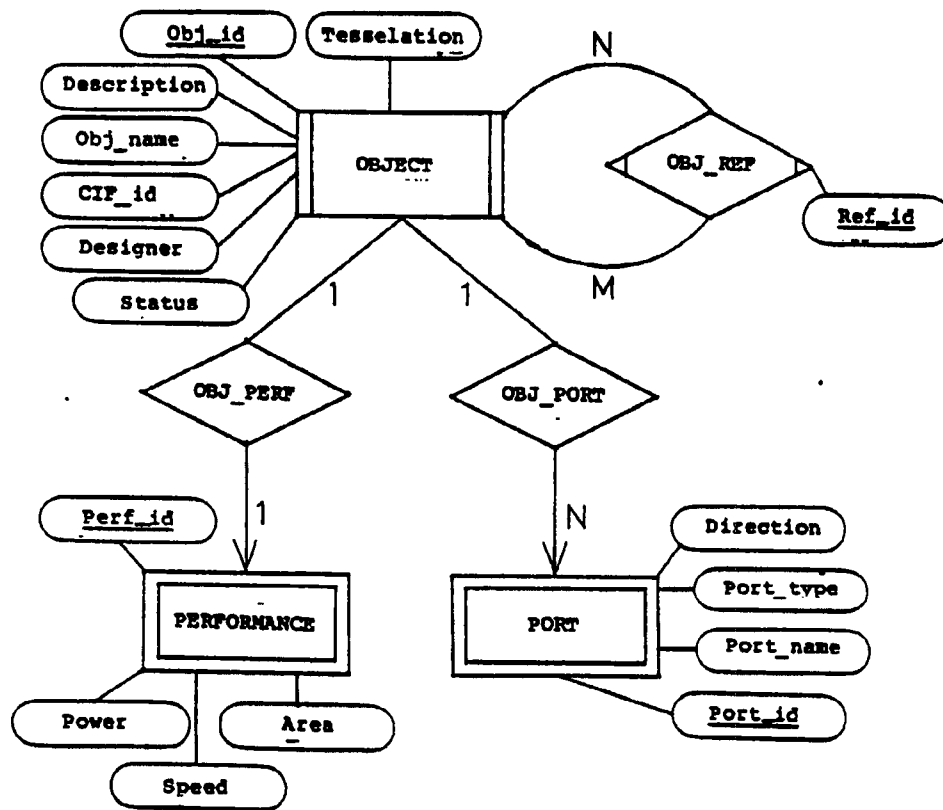


Figure 5. The Interface-Description View

1. Name

The name and version of the object must be indicated. Example is "FullAdder/1.1".

2. CIF id

The Caltech Intermediate Form is used as a standard layout-description format [10]. The CIF_id is the CIF symbol number and it uniquely identifies the cell from other cells, within the database of a design object. However, other design objects may use similar CIF_ids. So the cell_id, and not the CIF_id, uniquely identifies a design object in a common database.

3. Designer

The designer or group responsible for the object's implementation is identified.

4. Description/Function

The object's functional behavior is described in greater detail. This information is used for documentation purposes.

5. Graphic Representation

7. Performance Information

This information specifies constraints involving power, speed, and area.

8. Tessellation

This information comments on stacking and abutting of the object. For example, "abuts horizontally" means the object or cell can be placed side-by-side.

9. Status Information

Different categories are defined to standardize the state of each design as follows:

1. Checked. Design-rules-checks (DRC) and electrical simulation are required for all objects, as a minimum requirement.
2. Tested. The object has been extensively tested, and it works!
3. Iterated. It was tested, found to have bugs, and so was redesigned.
4. Generally Used. It was tested, and found to be working in at least three designs.

5. Pending. The object has undergone some minor modifications, and checks have been performed, but yet to be tested. Further actions are anticipated.

An interface description constrains the object's implementation. In order to connect design objects correctly, the database system has to check for any violations in the interface description of each design object. Checking that power and area are within specified constraints is relatively simple. Associated with the geometrical representation, area can be calculated from the bounding box of its geometries. Similarly, the dc power consumption can be determined from the width-to-length (gate) ratios of the transistors.

2.3.2 Geometry (or Layout) View

At the layout level in VLSI design, the designer is concerned with only the geometrical description of a design object. Hence, an individual view called the geometry or layout view is created. Figure 6 shows the geometry view of an object or cell based on the EE-R data model. The entity set "CELL" is of the kernel class, and "PART_OF" relationship set is of N:M reference relationship class. The geometrical structure of a cell is composed of primitives such as "BOXes", "WIRES", and "POLYGONS". Hence,

each of these entity sets belong to the owned class, and are connected to the owner "CELL" by 1:N relationships. These 1:N binary relationships are weak relationships.

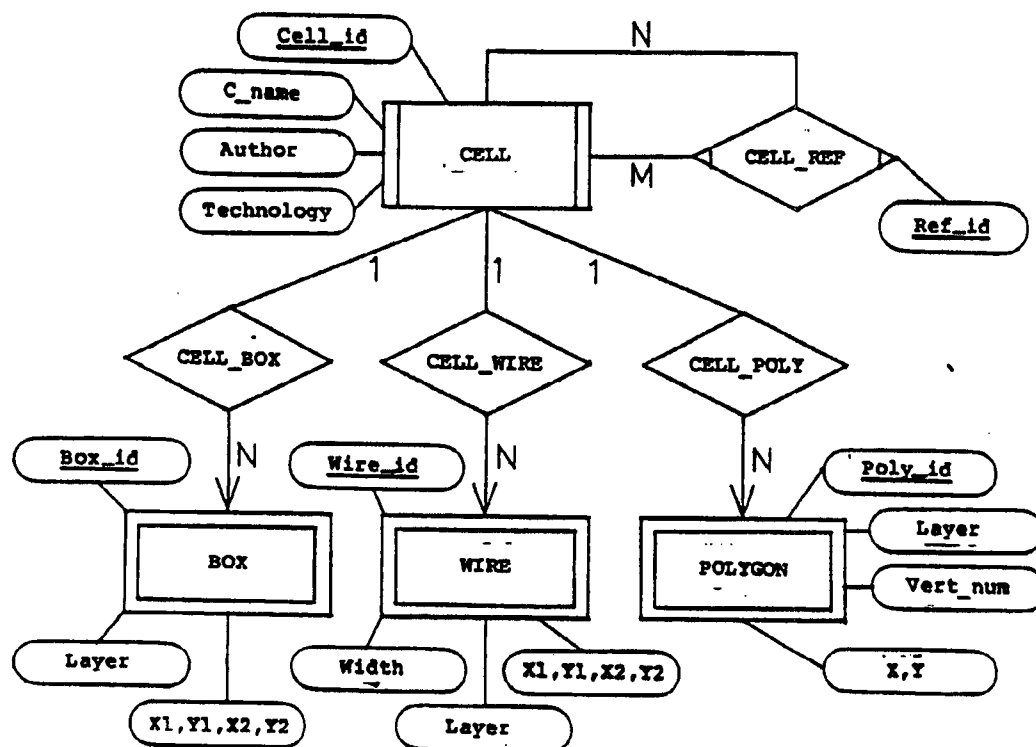


Figure 6. The Geometry View

-27-

CHAPTER III

CONCEPTUAL SCHEMA

As part of the logical database design process, the information represented by the enterprise model needs to be mapped into a conceptual schema that can be handled by the database management system. The conversion, from the EE-R data model to a relational data model, results in a relation schema or a set of tables.

3.1 The Relational Data Model (RDM)

The algorithm for the E-R data model to RDM conversion can be extended to include the entity classes and relationship classes defined in the EE-R data model. The conversion process can be summarized as follows:

1. KERNEL ENTITY SET

A kernel entity set can be represented by a KERNEL ENTITY RELATION whose relation schema consists of all the attributes of the kernel entity set. Each tuple in the kernel entity set represents a design object. The primary key of the kernel entity relation is equal to the key of the kernel entity set.

2. OWNED ENTITY SET

An owned entity set can be represented by an OWNED ENTITY RELATION whose relation schema consists of all the attributes of the owned entity set plus the key attributes of the supporting kernel entity sets. The primary key of the owned entity relation is formed by the concatenation of the partial key of the owned entity set and the keys of the supporting entity sets.

3. REFERENCE RELATIONSHIP SET WITH ATTRIBUTES

A binary relationship set with attributes can be represented by a REFERENCE RELATIONSHIP RELATION whose relation schema consists of all the attributes of the reference relationship set plus the key attributes of the two supporting kernel entity sets. The primary key of the relationship relation is

1. formed by the concatenation of the two kernel entity set keys, if the mapping class is M:N
2. equal to the key of the target kernel entity set, if the mapping class is N:1 or 1:N
3. equal to either of the two keys of the supporting kernel entity sets, if the mapping class is 1:1

4. BINARY RELATIONSHIP SET WITHOUT ATTRIBUTES

A binary relationship set without attributes will be represented by a RELATIONSHIP RELATION only if it is of the mapping class M:N. In this case the relation schema consists of the key attributes of the two entity sets and the primary key is formed by the concatenation of the two entity set keys.

In the case of a weak binary relationship set of mapping class 1:N or N:1, the primary key of the target entity set is formed by the concatenation of key attributes of the source entity set and the key attributes of the target entity set.

If the binary relationship set is of the mapping class 1:1, then associate the key of one entity set as a nonkey attribute of the other entity relation, and vice-versa. The reason is that if the relationship set is of mapping class 1:1, the two entity sets are both source and target entity sets.

The individual results of view modeling can be used to design each user schema, while the result of view integration (the enterprise schema) becomes, after conversion, the conceptual schema, and is represented by the relational data model. For each user schema, view relations (tables) are built.

Following the above conversion algorithm, the view relations for the geometry view of a design object are defined as follows:

LAYOUT_CELL (cell_id, cell_name, designer, technology)

LAYOUT_CELL_REF (parent, child, ref_id)

BOX (box_id, owner, layer, x1, y1, x2, y2)

WIRE (wire_id, owner, layer, x1, y1, x2, y2, width)

POLYGON (owner, polygon_id, layer, vert_num, x, y)

In the LAYOUT_CELL relation, cell_id is the primary key or a unique identification number assigned to each cell. It is used for unambiguous references to the cell within the common database. Cell_name is the textual name assigned to the cell, and designer is the name of the person or group responsible for the cell design. Technology is the fabrication process used, such as "NMOS 2.5" process, etc.

Each LAYOUT_CELL_REF describes the use of a particular cell as part of another, i.e., as a subcell. The parent field contains the identifier of the composite cell, and the child field contains the identifier of the subcell. Note that the cell_ref relationship cannot be cyclic.

The BOX relation describes the mask rectangles for mask-making. Box_id is a unique identification tag assigned to each BOX tuple within a particular cell. Owner is the identifier of the cell of which the box is a part. Layer specifies the processing layer, e.g., "polysilicon",

"diffusion", and so forth. The concatenation of box id and owner forms the primary key, and it uniquely identifies the box in the design database. The attributes x1, y1, x2, and y2 denote the coordinates of the bottom and top corners of a box.

A "wire" serves as an electrical connection between different terminals of a circuit. Each tuple in the WIRE relation describes one line segment, giving the coordinates of its centerline and its width. Wire_id is a unique identifier for a particular wire. The concatenation of the wire_id and the owner uniquely identifies the wire in the common design database.

Polygons are closed figures with at least three vertices. Each vertex is stored in each tuple of the VERTICES relation. A set of these tuples describes a polygon. X and y are the coordinates of the vertex, and vert_num orders the vertices (tuples) within one polygon. Note that the concatenation of the polygon_id and the owner uniquely identifies a particular polygon.

The relation schema for the interface-description view of a design object consists of four main relations defined as follows:

INTER_DESC_CELL (cell_id, cell_name, designer, CIF_id,
function_desc, status, tessellation, perf_id)

INTER_DESC_CELL_REF (parent, child, ref_id)

- 32 -

PERFORMANCE (perf_id, owner, power, speed, area)

PORT (port_id, owner, port_name, port_type, direction)

The attributes of INTER_DESC_CELL relation are defined in the enterprise schema discussed earlier. The PERFORMANCE relation specifies the power, speed, and chip area characteristics of the design object. The PORT relation constrains the connectivity of the design object to the "outside world".

The integration of the above geometry and interface-description relation schemas forms the relational data model for a design object. Redundancies are removed, and any ambiguities resolved, before the user schemas are integrated into one comprehensive relational data model. The result is a conceptual schema consisting of eight main relations as follows:

CELL (cell_id, cell_name, designer, CIF_id, status,
function_desc, tessellation, technology, perf_id)

CELL_REF (parent, child, ref_id)

BOX (box_id, owner, layer, x1, y1, x2, y2)

WIRE (wire_id, owner, layer, x1, y1, x2, y2, width)

POLYGON (owner, polygon_id, layer, vertnum, x, y)

PERFORMANCE (perf_id, owner, power, speed, area)

PORT (port_id, owner, port_name, port_type,
direction)

Notice the LAYOUT_CELL and INTER_DESC_CELL relations have been replaced by a common CELL relation.

- 33 -

3.2 Normalization

Two kinds of restrictions are placed on the relational data model. They are

1. Restrictions that depend on the semantics of the domain elements. These restrictions depend on understanding what components of tuples mean. Examples of such restrictions are

* No man has height > 10 feet.

* Salary cannot be < 0, etc.

It is important that the DBMS check out such plausible values if the database is to be meaningful at all. These restrictions called integrity constraints are concerned with the actual values stored in a database.

2. Restrictions on relations that depend only on the equality or inequality of values. These constraints, called functional dependencies, do not depend on what value a tuple has in any given component, but only on whether two tuples agree in certain components.

Definition: Given a relation R, attribute Y of R is Functionally Dependent on attribute X of R iff, whenever two tuples of R agree on their X-value, they also agree on the Y-value, i.e.

$X \twoheadrightarrow Y$ (X identifies Y)

1NF (First Normal Form)

Under 1NF, all occurrences of a tuple must contain simple domains, i.e., the 1NF excludes repeating groups. Removing all nonsimple domains results in relations of the form of flat files. Relations in their 1NF are called normalized relations. Theoretically, relational database system does not support relations with complex or nonsimple domains. Normalized relations are favored so that the resulting simplifications in data structures leads to corresponding simplifications in the storage and manipulation of design data.

In this research, the issues of functional dependencies in the relation schema have been neglected, and all relations are assumed to be in their 1NF. The following section discusses the restrictions placed on the actual values of tuples stored in a design database.

3.2.1 Enforcing Integrity Constraints on Design Data

Data integrity means that data in a given table conforms to specifications established by the user. Data integrity requires that the contents of the table conform to user's specifications, just as the format of a table conforms to a given specification. Enforcing integrity constraints is the mechanism for protecting the database from unacceptable updates.

To enforce constraints on design data, a user's query is modified so that the following can occur:

1. If the query is an update operation, it is executed only if no pre-declared integrity constraint is violated.
2. If the query is a retrieval, it is executed so that only the data the user is authorized to see are retrieved.
3. If the query is on user-defined view tables, the modified query is executed on real tables.

In CAD applications, complying with the design rules are examples of such restrictions:

- * No box has width or length ≤ 0

- * No performance has speed or area ≤ 0

or we may want to place a range, such as

- * No polygon has number of vertices ≥ 20

In QUEL, integrity constraints can be placed on a table using the "define integrity" command. Unfortunately, QUEL does not support multiple variable qualification or aggregates when defining an integrity constraint on a table. However, this problem can be solved by creating view tables from the base tables. Unlike base tables, view tables are calculated and are not stored permanently in the disk. For example, we can define a view table

```
prim_box (owner, box_id, layer, width, height)
```

where width and height are calculated from (x2-x1) and (y2-y1), respectively, of the base table "BOX". Hence, we can defined the above restrictions as follows:

```
* define integrity on b is b.width > 0
* define integrity on b is b.height > 0
* define integrity on s is s.speed > 0
* define integrity on s is s.area > 0
* define integrity on p is p.vert_num <= 20
```

Updates that violate any integrity constraints are simply ignored by the system.

CHAPTER IV

RETRIEVING DESIGN INFORMATION

Application programs can be written to manipulate the design database via the data manipulation or query language of a host database system.

4.1 QUEL (QUERY Language)

QUEL is INGRES' query language based on predicate calculus, which allows the user to retrieve, manage, and maintain data in an existing INGRES database. Queries are issued to INGRES using relational operators, aggregate functions, or a combination of both.

4.1.1 Relational Operators

The basic relational operators in QUEL are based on relational algebra. The most familiar relational operations are:

- o Projection
- o Restriction
- o Cartesian Product
- o Join

Projection

Projection is an operator that constructs a "vertical section" of an existing table by taking a subset of its columns. Suppose the design database has two view relations as follows:

design (cellid, cellname, author, area, status)

primBox (owner, layer, length, height, cx, cy)

Then the theoretical statement

project design on (cellname, area)

specifies a table consisting of the "cellname" and "area" columns of the "design" table. The "target list" of a QUEL statement corresponds to projection. Therefore, the above theoretical statement can be expressed in QUEL as

range of c is design

retrieve (c.cellname, c.area)

Restriction

Restriction (or selection) is used to construct a "horizontal section" of a table by taking those tuples that satisfy a specified condition. For example, the theoretical statement

restrict design on (cellid > 100)

defines a table consisting of all tuples in the "design"

relation for which the value in "cellid" is greater than 100. The "where clause" of a QUEL statement corresponds to restriction. Hence the above theoretical statement is expressed in QUEL as

```
retrieve (c.all) where c.cellid > 100
```

Cartesian Product

The cartesian product of two tables, say A and B, is a table (A*B) consisting of all concatenations of tuples from A with tuples from B. For example, "design*primBox" is a table consisting of all concatenations d*b, where d is a tuple from the "design" relation and b a tuple from the "primBox" relation. The range variables of QUEL provide a perfect means for expressing the cartesian product. For example, the theoretical "design*primBox" is expressed as

```
retrieve (d.all, b.all)
```

Join

The "join" operator constructs a table out of two existing tables by collecting all pairs of tuples such that each pair satisfies some condition. When the condition is equality between columns from the tuples, the operator is called an "equijoin". For example, the theoretical statement

join design with primBox on
 (cellid of design = owner of primBox)

is an equijoin. By contrast, the following statement

join design with primBox on
 (10*(area of design) < length of primBox)

is a join but not an equijoin.

A join is equivalent to a combination of a cartesian product followed by a restriction. For example, the above second join is equivalent to the formulation

restrict (design*primBox) on
 (10*(area of design) < length of primBox)

Then the above examples of equijoin and join can be expressed in QUEL as follows:

retrieve (d.all, b.all) where d.cellid = b.owner
 retrieve (d.all, b.all) where 10*d.area < b.length

Combination of relational operators

In QUEL, it is possible to combine a succession of relational operators in a single statement. Consider the following theoretical statement

Get all names of designers who have designs of
 area greater than 1000 and with length

of box less than 20

In QUEL, the above query can be expressed as:

range of d is design

range of b is primBox

retrieve unique (d.author) where d.area > 1000

and d.cellid = b.owner and b.length < 20

4.1.2 Aggregate Functions

Aggregation refers to operations that derive a single value from a group of data items. The operations can derive the single value either by specific computations or by evaluating the group of data items according to specific tests. INGRES supports aggregate operators that allow the user to perform statistics on all or some values in a column. Some examples of common aggregate operators are

```

avg      -- compute average value of elements in group
avgu     -- compute average value of unique elements
           in group
count    -- number of elements in group
countu   -- number of unique elements in group
sum      -- compute total value of elements in group
sumu     -- compute total value of unique elements in
           group
max      -- maximum value
min      -- minimum value

```

Suppose there is a view table named "design" which contains the following tuples:

design			
cellid	cellname	area	status
90	3-T-RAM	200	tested
121	plain	2500	checked
326	plaout	3000	checked

Then

```

sum(c.area where c.cellid > 100)

```

evaluates to 5500 micron-square.

4.2 Applications of Abstract Data Types

This section discusses the mechanism of adding new data types and new operators to a relational DBMS for CAD applications. Using a form of abstract data types (ADT), new data types for columns of a relation, such as boxes, wires and polygons become possible.

Circuit designs are hierarchical; a complete design expands into a tree, with a single cell at the root and instances of subcells for the non-root nodes. Each cell can contain mask geometry and subcell references. Registers, buffers, NOR gates, etc. can be classified as cells.

Guttman and Stonebraker [3] described a relational schema consisting of five main relations:

```
cell_master (name, author, master_id)
box (owner, layer, x1, x2, y1, y2)
wire (owner, layer, wire_id, width, x1, y1, x2, y2)
polygon (owner, layer, polygon_id, vertnum, x, y)
cell_ref (parent, child, cell_ref_id, t11-t32)
```

In the cell_master relation, name is the textual name given to the cell and author is the name of the designer. Master_id is a unique identification number assigned to each cell. It is used for unambiguous references to the cell within the database.

The box relation describes mask rectangles. Owner is the identifier of the cell of which the box is a part. Layer specifies the processing layer, e.g., "polysilicon", "diffusion", etc. X1 and x2 are the x coordinates of the left and right sides of the box, while y1 and y2 are the y_coordinates of the top and bottom.

A "wire" is a set of lines that serves to make an electrical connection between different parts of a circuit. Each tuple in the wire relation describes one line segment, giving the coordinates of its centerline (x1, y1, x2, y2) and its width. Wire_id is a unique identifier for a particular wire.

A polygon is a closed figure with any number of vertices. One vertex is stored in each tuple of the polygon relation. X and y are the coordinates of the vertex, and vertnum orders the vertices (tuples) within one polygon.

Each cell_ref tuple describes the use of one cell as part of another, i.e. as a subcell. For example, the cell REGISTER contains several LATCH subcells. Then, there would be several cell_ref tuples, each containing the identifier of the REGISTER cell in the parent field, and the identifier of the LATCH cell in the child field. Note that the cell_ref relationship cannot be cyclic, i.e., LATCH cell cannot be made the parent field when REGISTER cell is a child field. T11 through T32 is a 3X2 matrix of floating

point numbers specifying the location, orientation and scale of each subcell with respect to its parent.

With the definition of abstract data types "box ADT", "polygon_ADT", "wire_ADT" and "array of floats", we can simplify the above schema to:

```
cell_master (name, author, master_id)
box (owner, layer, box_desc)
wire (owner, layer, wire_desc)
polygon (owner, layer, polygon_desc)
cell_ref (parent, child, cell_ref_id, orientation)
```

In CAD applications, objects are often made up of "boxes". For a VLSI database, one can define a "boxes" relation as follows:

```
* create boxes (owner=i4,
                layer=c15,
                box_desc=box_ADT)
```

The boxes relation has three fields: the identifier of the circuit of which each box is a part, the processing layer for the box (polysilicon, diffusion, etc.) and a description of the box's geometry. All fields are represented by standard built-in types except box_desc which is a data type declared by the ADT implementor.

Tuples can be added to the BOXES relation using QUEL as follows:

```
* append to boxes (owner=99,
                    layer="polysilicon",
                    box_desc="0,0,2,3")
```

An input procedure must be available for the DBMS to perform the conversion of character string "0,0,2,3" to an object with data type box_ADT.

One can also use standard DBMS operators on the box ADT domain, e.g.,

```
* range of b is boxes
* replace b (box_desc=b.box_desc * "0,0,4,1)
  where b.owner=99
```

This above command will replace the box by its intersection with another box. In this case "0,0,4,1" will be converted to an object of type box_ADT to match the type of b.box_desc, and then the appropriate multiplication is performed.

Similarly, one might want to define a function that calculates the area of a box and use it in data manipulation commands, e.g.,

```
* range of b is boxes
* retrieve (b.owner)
```

where area(b.box_desc) > 100

Here one must define the function "area" which accepts a box_ADT as an input and returns an integer.

In addition, one can define new comparison operations. With the operator "||" defined for "overlaps" of boxes, we can query if there were any boxes overlapping the unit square based at the origin as follows:

```
* range of b is boxes
* retrieve (b.box_desc)
  where b.box_desc || "0,0,1,1"
```

Again, a procedure for the "||" operator has to be predefined by the ADT implementor.

Lastly, one can also define aggregate functions for this new data type. For example, assuming the "tallest" function is predefined, the command to find the owner of the box with the largest vertical dimension on the polysilicon layer would be:

```
* range of b is boxes
* retrieve (b.owner) where b.box_desc=
  tallest(b.box_desc where b.layer="polysilicon")
```

Assuming the availability of the ADT implementor, consider the following example relation schema:

```

cell (name, author, status, cell_id,
      alternative_id)
box (owner, layer, box_desc)
wire (owner, layer, wire_desc)
polygon (owner, layer, polygon_desc)
part_of (parent, child, part_of_cell_id,
         tessellation)

```

In the cell relation, "name" is the textual name of the circuit, e.g., PLA. Author is the designer responsible for the cell design. Status gives information on whether the cell has been tested, is used frequently or has bugs. Cell_id is a unique identification number assigned to each cell in the library. Alternative_id is the identification number of a different implementation of the same function but with varying performance such as speed and power consumption. Size of the cell can be calculated by "pulling" out all primitives with the same owner, and calculating its boundaries.

In the part_of relation, tessellation describes how the cell abuts to its neighbors such as overlaps. The other relations have the same definitions as described previously.

With the use of ADTs, we can simplify the cell relation to:

```

cell (name, author, status, cell_desc)

```

Then we can define a "cells" relation as follows:

```
* create cells (name=c16,
                author=c16,
                status=c16,
                cell_desc=cell_ADT)
```

where "cell_ADT" is predefined by the ADT implementor.
Tuples can be added to the "cells" relation by command:

```
* append to cells (name="3x8x2-PROM",
                  author="simon foo"
                  status="checked"
                  cell_desc="100,76")
```

where the string "100, 76" specifies the cell_id and the alternative_id, respectively. A programmable read-only-memory (PROM) is a special type of PLA with a "fixed" AND plane and a programmable OR plane.

Similarly, to find the author of a particular RAM cell, we can write a simple query:

```
* range of c is cells
* retrieve (c.author) where
    c.name="RAM", c.status="tested",
    c.cell_desc="99,88"
```

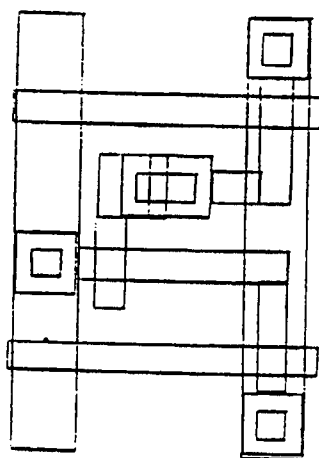


Figure 7. A 3-Transistor dynamic RAM cell

Figure 7 shows the physical layout of a 3-transistor dynamic RAM cell, where "boxes" represent different processing layers.

At a higher level of circuit representation, we can add new relations such as:

```
network (net_id, author, net_fcn, net_type, x1, x2)
channel (owner, channel_num, num_terminals, num_tracks)
```

Carry-save-adder (CSA) networks are commonly used in the design of numeric processors. In the network relation, net_fcn specifies the function of the network, while net_type is the implementation of the circuit. X1 and x2 represent the size of the operands or inputs to the network. Routing channels shows the interconnections between two rows of cells in a CSA network. In the channel relation, owner is the identifier of network which it belongs to, and channel_num specifies the order of the routed channels. Num_terminals and num_tracks specify the number of connecting points and the number of horizontal tracks required for routing, respectively.

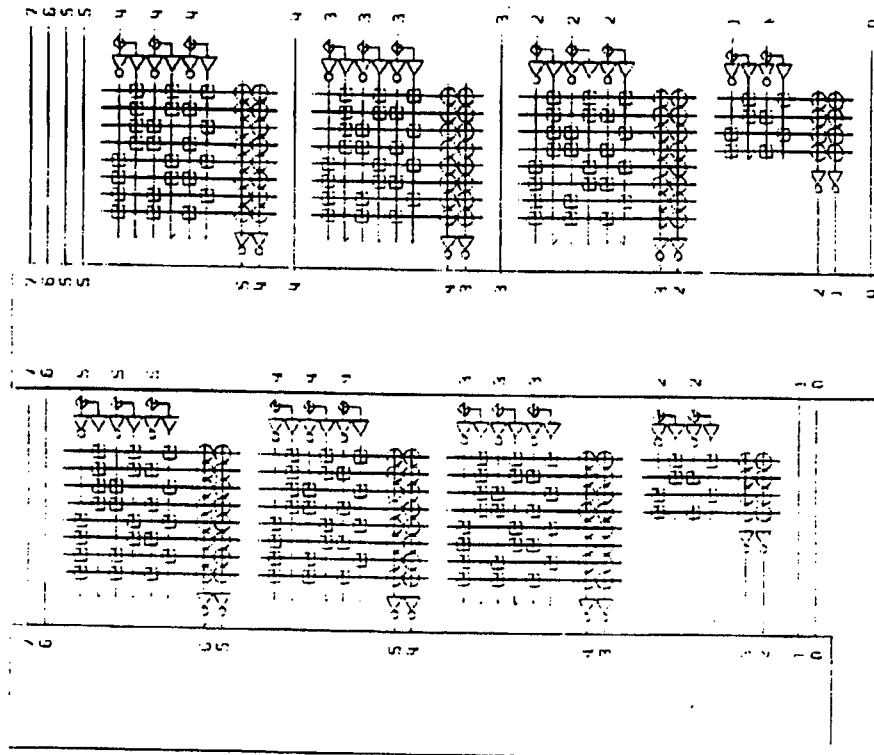


Figure 8. Logic schematic for a 4x4 CSA network

Similarly, with definition of abstract data types "net_ADT" and "channel_ADT", we can simplify the above schema to

```
* network (net_id, author, net_fcn,
           net_type, net_desc)
* channel (owner, ch_desc)
```

Then we can write a simple query such as

```
* range of n is networks
* retrieve (n.author) where
           n.net_fcn="CSA", n.net_type="PLA",
           n.net_desc="4,4"
```

Figure 8 shows a logic schematic of a CSA network for 4X4 multiplication. Notice that the network is synthesized using PLA cells defined earlier as building blocks. The "box" between the two horizontal rows of cells represent the routing channel.

The above examples clearly show the usefulness of ADTs in relational DBMS for CAD applications. Relational operators and aggregate functions can also be predefined for this CAD system. Queries are greatly simplified as a result of the use of ADTs. Performance improvement from the use of ADTs can be summarized as follows:

1. Manipulation of a smaller number of columns. For example, a box can be retrieved as a single column rather than as four constituent parts.
2. Manipulation of fewer tuples for polygons represented by single fixed-size tuples with vertex lists stored externally.
3. Simplification of queries due to the introduction of new operators.

CHAPTER V

AN INTEGRATED CAD SYSTEM

A collection of CAD tools forms an integrated CAD system, with a relational DBMS as the heart of the system. The novel architecture of such a system is shown in Figure 9, where CAD tools are attached to the database system via interfaces (not shown in picture).

5.1 A Relational DBMS: INGRES

The primary task of the database system in a CAD environment is to represent and manage information vital to VLSI designs on a common database. If the database is to be meaningful, then the DBMS must maintain the consistencies of the design information across various levels of abstraction. INGRES, a commercial DBMS based on the relational data model, is used as the database system in this research. QUEL is the data manipulation language that allows users to manipulate the design data in a common database. Application programs can be written using the embedded form of QUEL (EQUEL).

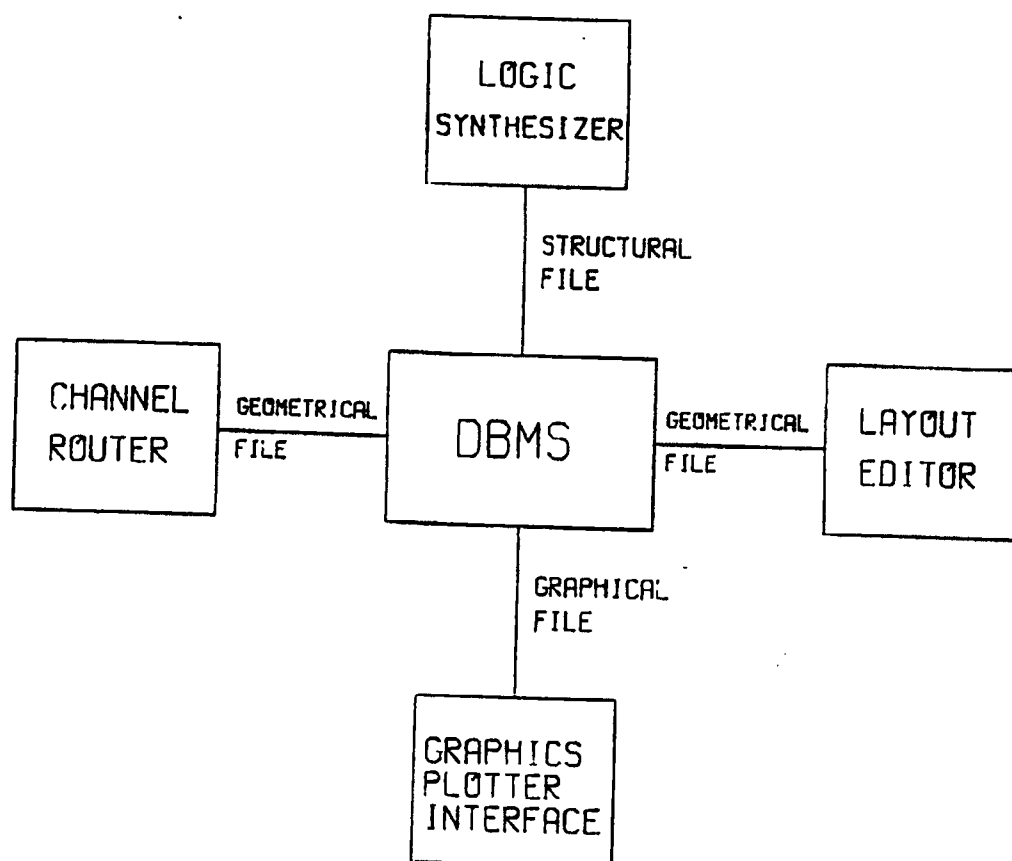


Figure 9. An Integrated CAD System

5.2 A Collection of CAD Tools

Essential CAD tools such as logic synthesizer , channel router , layout editor , and graphics interface have been developed, and are interfaced indirectly to the DBMS via some high-level and low-level intermediate files. A brief description of each module is presented as follows:

5.2.1 Logic Synthesizer: SPAN

SPAN (a Synthesis Program for Adder Networks) is a CAD tool for synthesizing logic networks of carry-save-adders (CSAs) [6]. Using both matrix-height reduction and carry-look-ahead minimization techniques, SPAN has been proven to yield better results than conventional approaches. SPAN is applicable for the design of special-purpose processors which require large adder networks. Standard output from SPAN is a structural file, specifying the primitive cells such as full-adders and half-adders that implement the CSA network for given input specifications. The file could be fed directly to channel routers, or to a graphics interface for human verification.

5.2.2 Channel Router: CHRP

CHRP (a CHannel Routing Program) is another CAD tool for routing interchangeable terminals [8]. Most channel routers assume that connecting terminals are rigid within each cell. However, in cases where programmable logic devices are used as building blocks, the terminals are "moveable" since the geometries are programmable. CHRP yields near optimum solutions for routing moveable terminals compared to other routing algorithms.

5.2.3 Graphics Interface: GILL

GILL is a graphical interface for displaying logic-level layout of custom adder networks on a color-graphics terminal or a plotter. This tool provides important visual aid to a user designing a custom logic network. GILL is capable of displaying the box-level simulation of a logic network, as well as generating the corresponding logic schematic using PLAs as building blocks. Figure 10 shows a box-level simulation of an example adder network.

5.2.4 Layout Editor: KIC

KIC is an interactive, color graphics editor for laying out integrated circuits [9]. KIC has symbolic layout and split-screen capabilities, and can handle layouts as large

as 200,000,000 lambda-square. KIC has a built-in dedicated database system, which is not transparent to the user. A KIC database consists of a collection of circuit "cells". Each cell contains the mask geometry and subcell references (if any). During editing, KIC stores the complete circuit in the virtual memory of a VAX-11/780 computer.

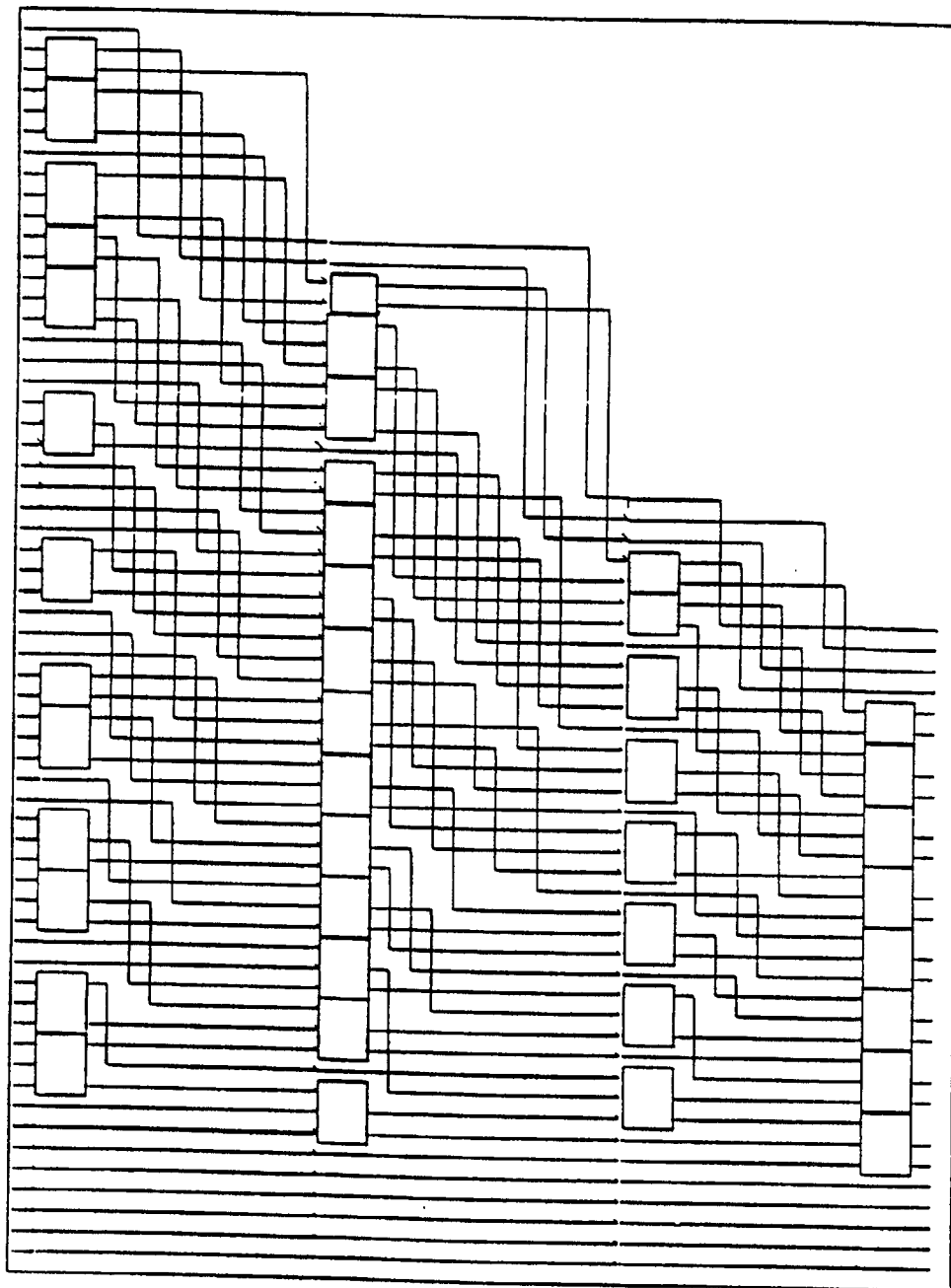


Figure 10. Box-level simulation of an adder network

CHAPTER VI

A VLSI DESIGN EXAMPLE

To evaluate the performance of the integrated CAD system, a design example of a parallel convolver using programmable logic is presented. The use of programmable logic for parallel convolution is provided as a background.

6.1 Programmable Logic for Parallel Convolution [4]

Introduction

Recent advances in VLSI technology have made it possible to integrate millions of transistors on a single chip. More powerful and inexpensive computing resources can be provided by VLSI. However, improvement in computing power by several orders of magnitude is beyond the capability of sequential von Neuman architectures. More efficient algorithms and advanced architectures using programmable logic are needed to extend computational capabilities of VLSI.

For reasons of regular structures and design flexibilities, programmable logic devices such as PROM's and FPLA's are often used to realize complex functions as lookup tables. For example, parallel convolvers for real-time signal and image processing can be realized by programmable

logic. However, the table lookup approach is not practical for large functions since the memory size required to store lookup tables increases exponentially with the address length.

This section describes an algorithm for partitioning filter functions into a set of smaller subfunctions. This leads to a novel architecture using programmable logic for parallel convolution. The relation between the memory size required to implement the parallel convolver and the number of partitionings is also discussed.

Function Partition

A digital nonrecursive filter (DNF) for Q data samples calculates the k -th output $Y(k)$ according to the expression

$$Y(k) = \sum_{q=0}^{Q-1} A(k-q) X(q)$$

where X 's and A 's are input variables and filter coefficients, respectively. Note that Q multiplications are involved in the convolution.

The idea of partitioning large functions into a sum of smaller subfunctions leads to significant memory savings [5]. Let the input variable $X(q)$ of n bits be partitioned into P suboperands of n/P bits each:

$$X(q) = \sum_{p=0}^{P-1} X_p(q) 2^{pn/P}$$

where $X_p(q)$ is the p -th suboperand. Then the filter function can be partitioned into a sum of P subfunctions:

$$Y(k) = \sum_{q=0}^{Q-1} \sum_{p=0}^{P-1} A(k-q) X_p(q) 2^{pn/P}$$

$$= \sum_{p=0}^{P-1} \left[\sum_{q=0}^{Q-1} A(k-q) X_p(q) \right] 2^{pn/P}$$

where the p -th subfunction is enclosed with square brackets. Figure 11 shows a tree expression of the p -th subfunction Y_p . Note that the filter coefficients A 's are not partitioned since they are constants which can be directly included in lookup tables. To utilize smaller memories, the input variables need to be partitioned into a larger number of suboperands.

Programmable Logic Architecture

Figure 12 shows a DNF architecture using a PMA and a parallel adder. The input variables of n bits are partitioned into P suboperands of n/P bit each. The current input is delayed by shift registers to yield the previous inputs. The lookup tables are calculated according to the filter specifications, and programmed into the memory array. Examples of PMAs for generating subfunction values are shown in Figure 13. The subfunction values are looked up

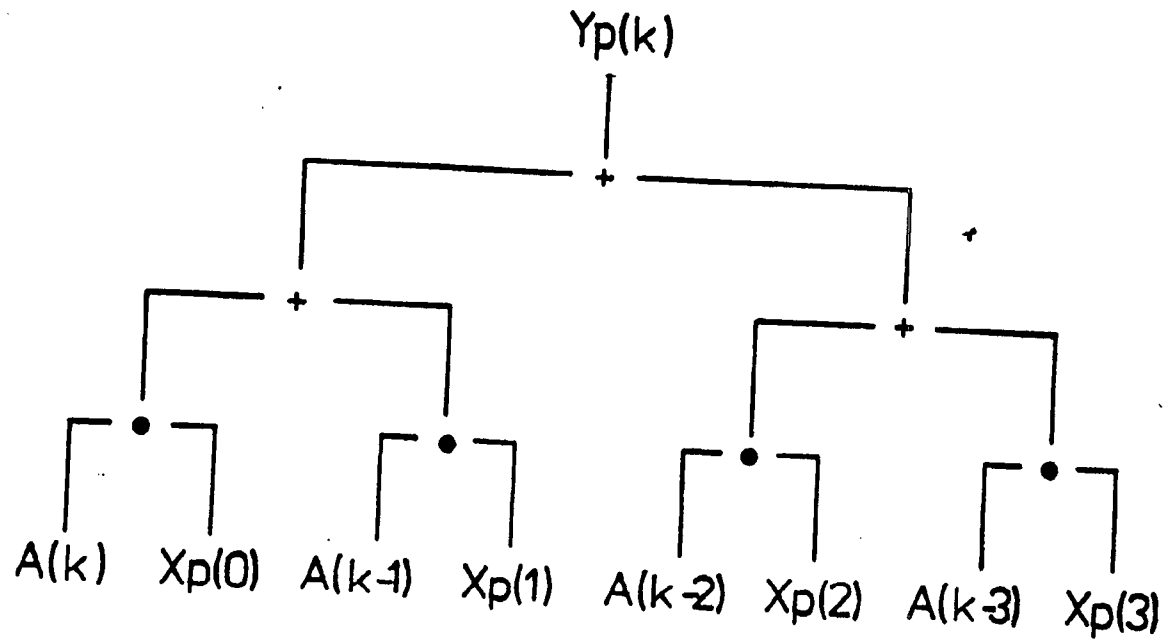


Figure 11. Tree expression of subfunction for $Q = 4$

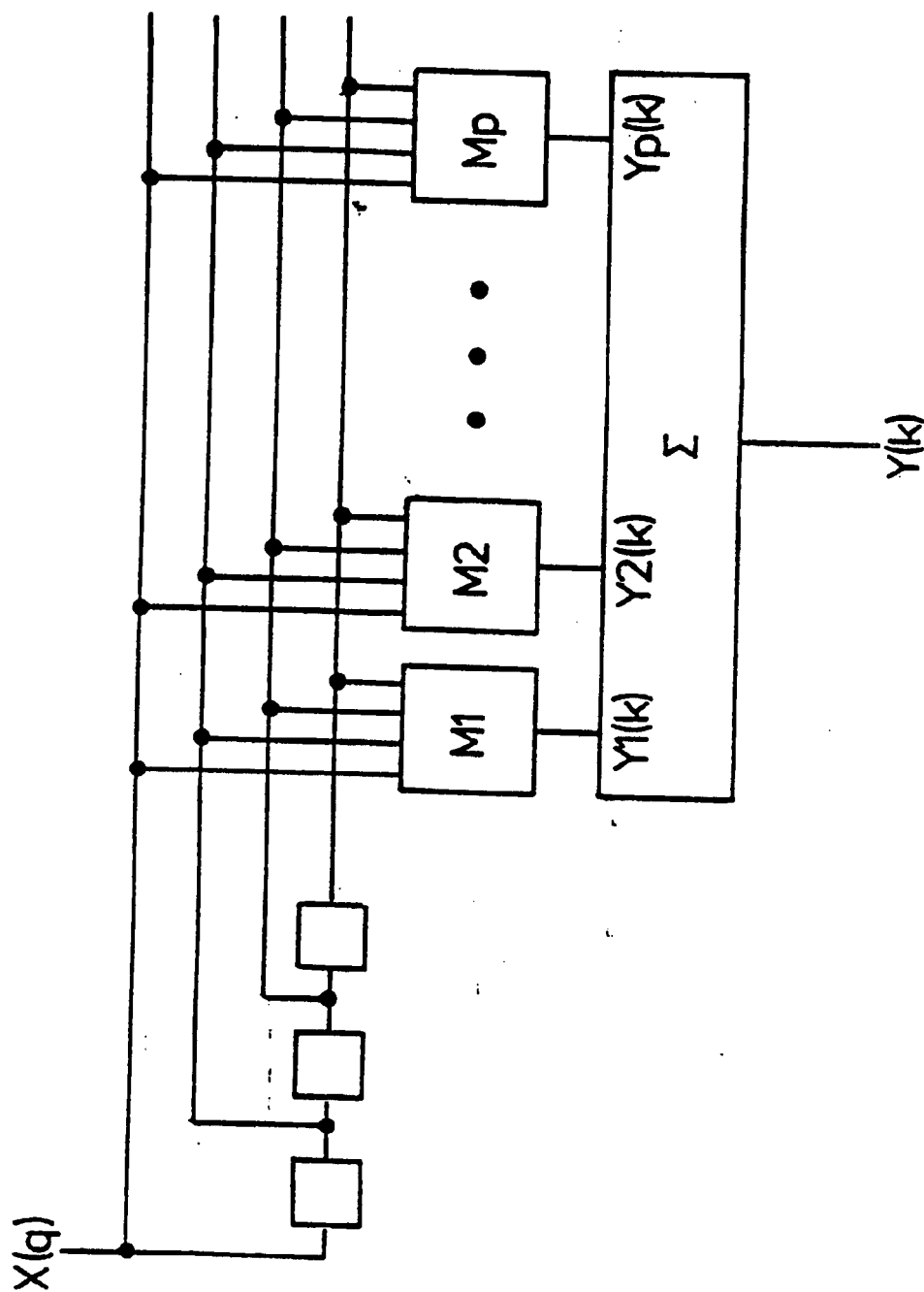


Figure 12. DNF architecture for $Q = 4$

simultaneously using the suboperands as addresses, then summed by the parallel adder to obtain the filter output. Note that the filter cycle time is only the memory access time and the parallel adder delay. Using current MOS technology, this leads to real-time processing for signals up to 10 MHz.

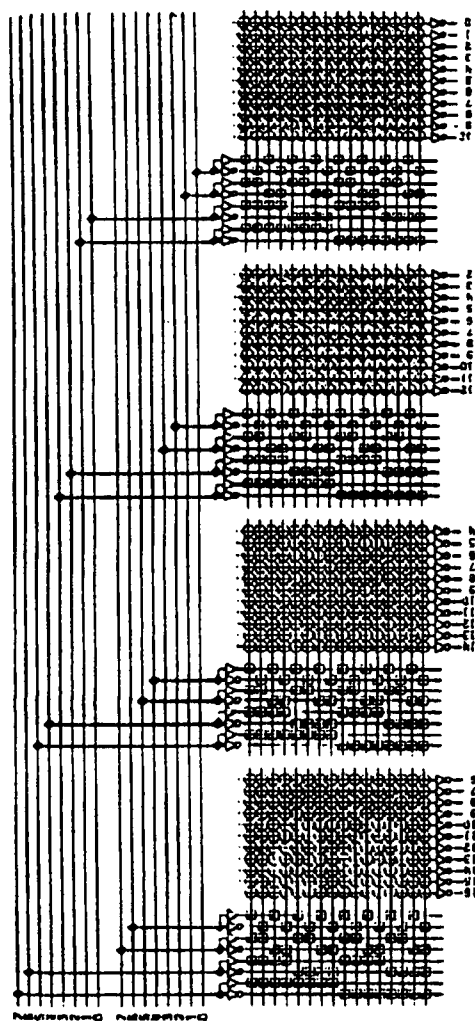
The subfunction values corresponding to Figure 13(a) are summed by a carry-save adder (CSA) network as shown in Figure 14, where boxes represent routing channels. PROM's are programmed with full and half adder tables. Finally, a carry look-ahead (CLA) adder is used to yield the filter output. Adder networks of this type can be synthesized by a CAD tool, SPAN [6].

Memory Size Versus Partitionings

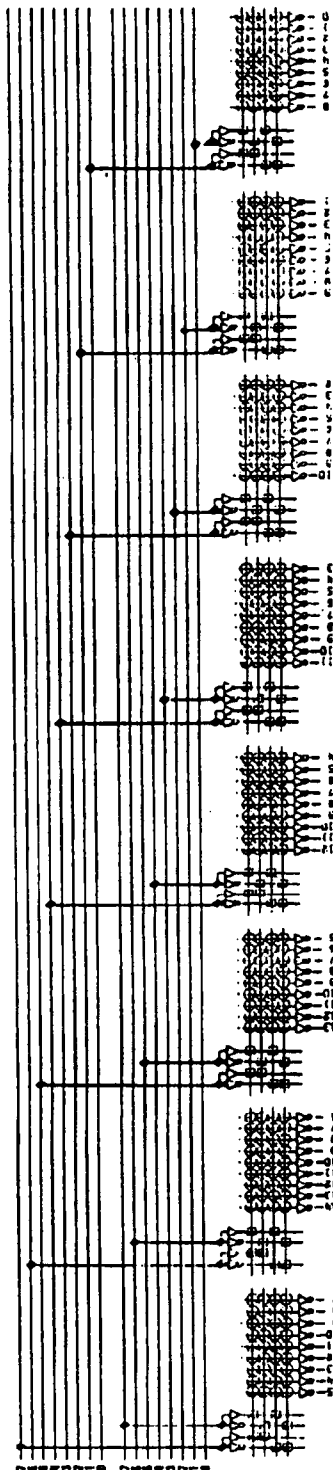
The memory address length, u , required to access the subfunction value depends on the number of bits in the suboperand and the number of data samples:

$$u = (n/P)Q$$

The data output length, v , required to represent the subfunction value depends on the number of bits in the coefficient and suboperand, and the logarithm of the number of data samples:



(a)



(b)

Figure 13. (a) PMA for $n = 8$, $p = 4$, and $Q = 2$
(b) PMA for $n = 8$, $p = 8$, and $Q = 2$

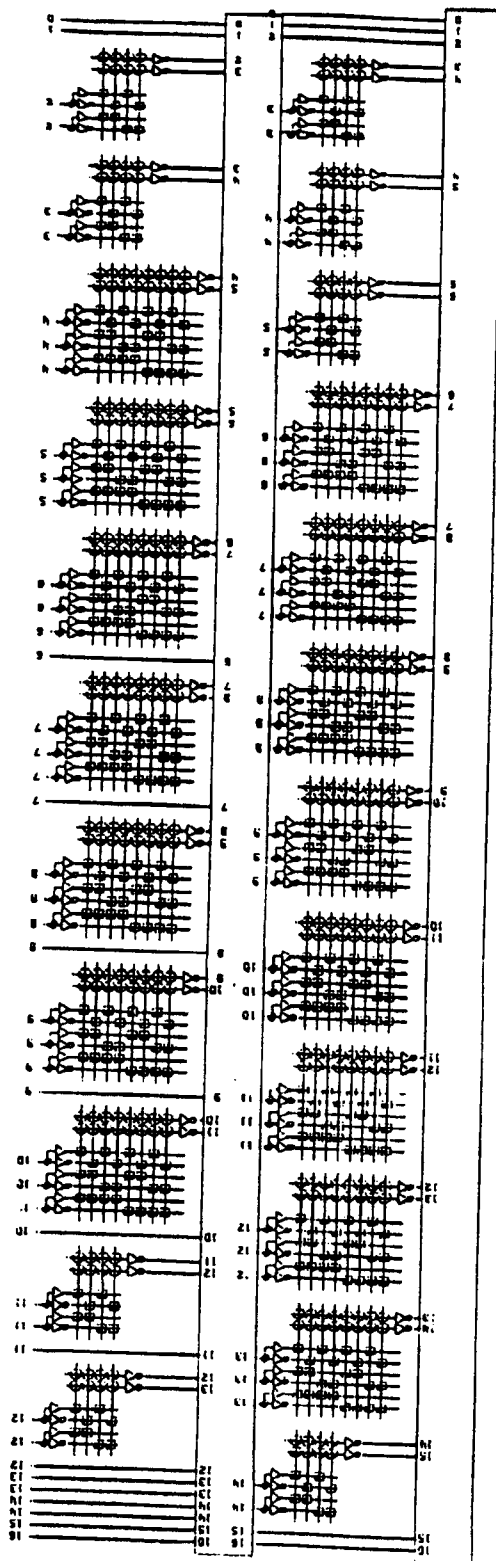


Figure 14. CSA network for $n = 8$, $p = 4$, and $Q = 2$

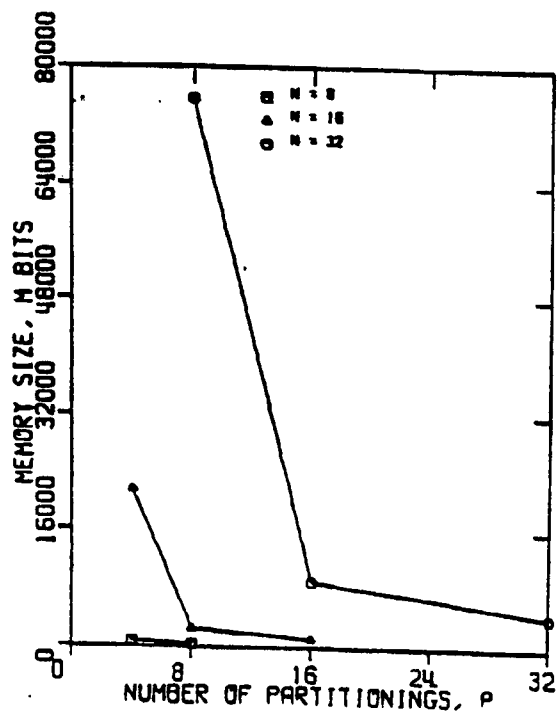
$$v = \begin{matrix} n + \lceil \log_2 Q \rceil & \text{for } n/P = 1 \\ n + n/P + \lceil \log_2 Q \rceil & \text{for } n/P > 2 \end{matrix}$$

where $\lceil X \rceil$ represents the smallest integer equal to or larger than X . Therefore, the number of memory bits, M , required to store P subfunctions is:

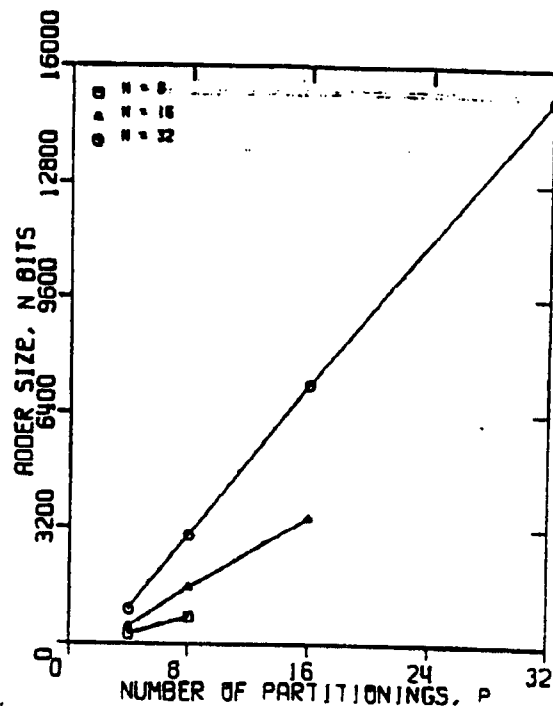
$$M = P(2^u . v)$$

The adder size required to sum P subfunctions increases with the logarithm of Q , while it increases linearly with Q in other parallel convolution schemes [7].

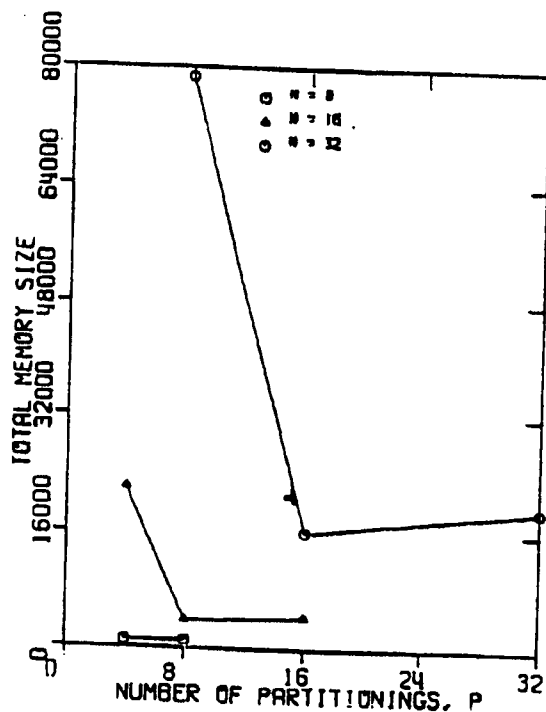
Figure 15(a) and (b) show the memory size M and the adder size N (in memory bits) versus P for $n=8, 16, 32$ and $Q=2$. Note that M decreases exponentially with P , while N increases linearly with P . The total number of memory bits required to implement a parallel convolver (except its CLA adder) is the sum of M and N . Figure 15(c) shows $(M + N)$ versus P for $n=8, 16, 32$ and $Q=2$. The results show that optimal partitioning can be achieved by $P=8$ and 16 for $n=16$ and 32 , respectively.



(a)



(b)



(c)

Figure 15. (a) Plot of M versus P for $n = 8, 16, 32$
 (b) Plot of N versus P for $n = 8, 16, 32$
 (c) Plot of $(M + N)$ versus P for $n = 8, 16, 32$

6.2 Implementation of the Prototype PMA422 Chip

PMA422, is the acronym for a programmable memory array with 4-bit operands where each operand is partitioned into 2 suboperands, and with 2 product-terms. This PMA422 is designed as a memory component of a parallel convolver. The goal of this prototype is to demonstrate the optimization of chip area, whereby large lookup tables are partitioned into a set of smaller subtables [4]. Following the design hierarchy discussed in Chapter II, the design steps toward the development of the PMA422 chip can be summarized as follows:

At the conceptual level, the specifications (such as n , P , Q) and requirements (such as chip area, total number of pins) of the convolver function are defined. Figure 16 shows the logic schematic of the PMA422 based on programmable logic architecture. Since the function ($S = C0*X1 + C1*X2$) is partitioned into two subfunctions, two PROMs are needed to encode the multiplication program table. Each PROM has four inputs, and seven outputs (six bits to represent the product, and an extra bit to represent the sum of the two products) to represent the partial product matrix.

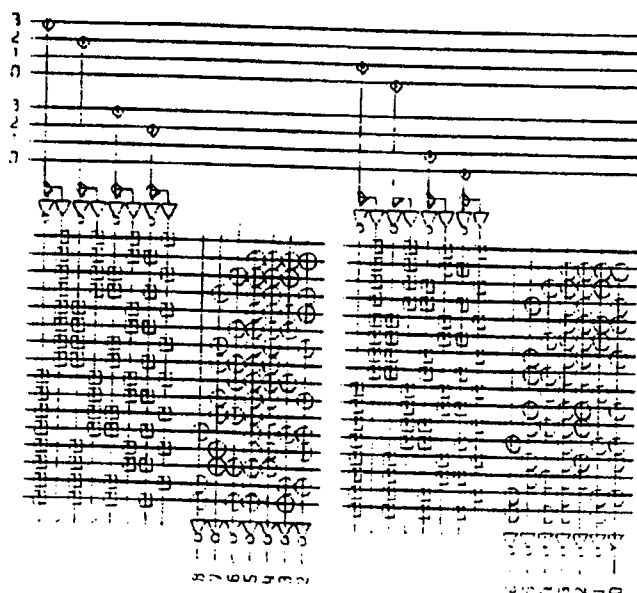


Figure 16. Logic Schematic of PMA422

To find the optimal partitioning of the PMA422, data are retrieved from the relational DBMS to make a decision on critical issues such as the memory area, the adder area, and hence the total chip area, depending on the nature of function partitioning. In addition, schematics such as Figure 16 are generated as a visual aid to the designer.

After all the necessary factors have been carefully analyzed, the next step is generating a multiplication program table for the given input function. Figure 17 shows the program table for the PMA422 chip. For testing purposes, F(hexadecimal) is used as an example coefficient.

The third step is to configure the floorplan of a PMA422 building block, pmaBlk. This composite object is a PROM composed of several PLA cells as shown in Figure 18.

Then, the design database is accessed to search for the availability of the needed standard PLA cells such as cellPair, bullupPair, etc. The information retrieved also includes the status of each cell. New cells are created (if needed), and are appended to the chip design database. The complete list of cells used in the PMA422, along with a brief description of each cell is given as follows:

Operand bit-length, n 4
 Number of partitions, p 2
 Number of terms, q 2

C0 := 1111 (decimal 15)
 C1 := 1111 (decimal 15)

Decimal

X1	X2	S := C0*X1 + C1*X2
0	0	0
0	1	15
0	2	30
0	3	45
1	0	15
1	1	30
1	2	45
1	3	60
2	0	30
2	1	45
2	2	60
2	3	75
3	0	45
3	1	60
3	2	75
3	3	90

Binary

X1	X2	S6	S5	S4	S3	S2	S1	S0
0 0	0 0	0	0	0	0	0	0	0
0 0	0 1	0	0	0	1	1	1	1
0 0	1 0	0	0	1	1	1	1	0
0 0	1 1	0	1	0	1	1	0	1
0 1	0 0	0	0	0	1	1	1	1
0 1	0 1	0	0	1	1	1	1	0
0 1	1 0	0	1	0	1	1	0	1
0 1	1 1	0	1	1	1	1	0	0
1 0	0 0	0	0	1	1	1	1	0
1 0	0 1	0	1	0	1	1	0	1
1 0	1 0	0	1	1	1	1	0	0
1 0	1 1	1	0	0	1	0	1	1
1 1	0 0	0	1	0	1	1	0	1
1 1	0 1	0	1	1	1	1	0	0
1 1	1 0	1	0	0	1	0	1	1
1 1	1 1	1	0	1	1	0	1	0

Figure 17. Program Table of PMA422

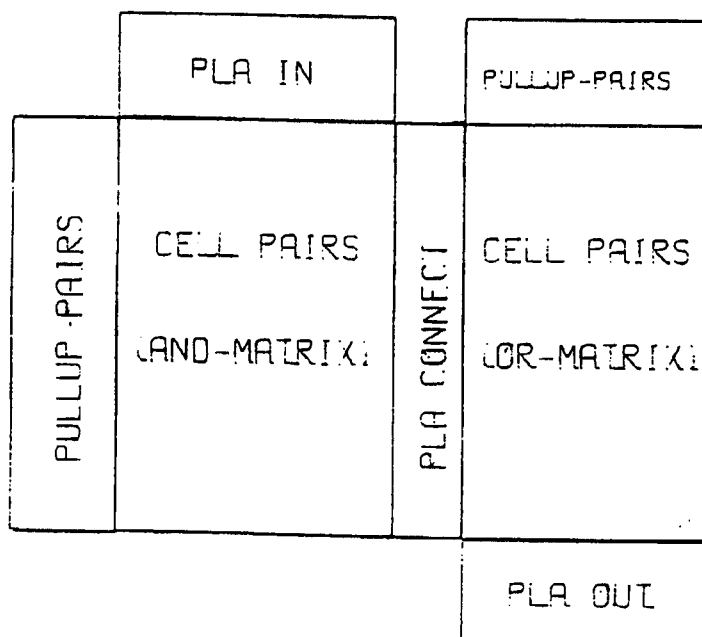


Figure 18. Floorplan of a pmaBlk

cellPair	-- A pair of "unprogrammed" transistors
padBlank	-- Core of a pad
padDriver	-- Interface circuitry for I/O ports
padGnd	-- Common ground pad
padIn	-- Input pad
padOut4	-- Output pad with 4:1 drive
padOut8	-- Output pad with 8:1 drive
padTstate	-- Tri-state pad
padVdd	-- Power supply pad
plaConnect	-- Connects the AND-plane to OR-plane of PLA
plaGnd	-- Common ground for PLA cells
plaIn	-- Circuitry for PLA input
plaOut	-- Circuitry for PLA output
pmaBlk	-- Main building block of PMA422
pullUpPair	-- Pair of N-channel enhancement-mode transistors

All the standard cells are described in Caltech Intermediate Form (CIF). The interactive KIC tool is used as the layout editor. The PMA422 is laid out by placing instances of the above cells at the proper place, according to the floorplan. The total chip area of the PMA422 chip is 1583X601 lambda-square. The complete design file for the PMA422 is the concatenation of all the individual files for the standard cells, and the listing is given in the Appendix.

The PMA422 design file is manually checked for any CIF violation before it is sent to MOSIS through the CSNET for fabrication. Upon receipt of the fabricated chips, the PMA422 chips will be tested for functional correctness, and design flaws (if any) will be detected.

6.3 Managing the Chip Design Database

The conceptual schema for PMA422 design data is based on the relational data model discussed in Chapter III. The relation schema consists of the following eight main relations:

CELL (cell_id, cell_name, designer, CIF_id, status, function_desc, tessellation, technology, perf_id)

CELL_REF (parent, child, ref_id)

BOX (box_id, owner, layer, x1, y1, x2, y2)

WIRE (wire_id, owner, layer, x1, y1, x2, y2, width)

POLYGON (owner, polygon_id, layer, vertnum, x, y)

PERFORMANCE (perf_id, owner, power, speed, area)

PORT (port_id, owner, port_name, port_type, direction)

An EQUOL/Pascal application program is written to retrieve, append, delete, and update the PMA422 design information. The above relation schema is created and defined in a common design database as follows:

```
* create cells (cell_id=i4,
                cell_name=cl6,
                designer=cl6,
                cif_id=i4,
                status=cl6,
                function_desc=cl6,
                tessellation=cl6,
                technology=cl6,
                perf_id=i4)

* create part_of (parent=i4,
                 child=i4,
                 ref_id=i4)

* create boxes (box_id=i4,
                owner=i4,
                layer=cl6,
```

- 78 -

```

        x1=i4,
        y1=i4,
        x2=i4,
        y2=i4)

* create wires (wire_id=i4,
               owner=i4,
               layer=c16,
               x1=i4,
               y1=i4,
               x2=i4,
               y2=i4,
               width=i2)

* create polygons (owner=i4,
                  polygon_id=i4,
                  layer=c16,
                  vert_num=i2,
                  x=i4,
                  y=i4)

* create cell_perf (perf_id=i4,
                   owner=i4,
                   power=i4,
                   speed=i4,
                   area=i4)

* create ports (port_id=i4,
               owner=i4,
               port_name=c16,
               port_type=c16,
               direction=c16)

```

Tuples can be appended to the "boxes" relation as shown below:

```

* append to boxes (box_id=236,
                  owner=99,
                  layer="diffusion",
                  x1=796,
                  y1=250,
                  x2=800,
                  y2=252)

```

Similarly, we can perform updates such as:

```

* replace c (status="tested")
  where c.author="simon foo"

```

which replaces the status of all cells designed by "simon foo" with status "tested".

Using relational operators, example queries are demonstrated below:

" Find all cell_names with author "simon foo"
 where status is "checked" and
 the technology used is "NMOS/2.5" "

In QUEL, the query can be expressed as

```
* range of c is cells
* retrieve (c.cell_name) where
    c.author = "simon foo" and
    c.status = "checked" and
    c.technology = "NMOS/2.5"
```

" Get all CIF_ids of cells with speed less than 100 ns "

The corresponding QUEL expression is

```
* range of p is cell_perf
* retrieve (c.cif_id) where
    p.owner = c.cell_id and
    p.speed < 100
```

" Get a list of all subcells belonging to cell "pmaBlk" "

Then the QUEL expression is

```
* range of t is cells
* range of p is part of
* retrieve (t.cell_id) where
    c.cell_name = "pmaBlk"
* retrieve (p.child) where
    p.parent = t.cell_id
* retrieve unique (c.cell_name) where
    c.cell_id = p.child
```

" Get a list of all NMOS cells' functional descriptions
sorted in alphabetical order "

The corresponding QUEL expression is

```
* retrieve unique (c.cell_name, c.functional_desc)
  where c.technology="NMOS"
  sort by cell_name
```

The above command will sort the tuples in ascending
alphabetical order, and all duplicate rows are removed
before being displayed.

CHAPTER VII
CONCLUSIONS AND DIRECTIONS
FOR FUTURE RESEARCH

The issues and processes of representing VLSI design data on a common database have been described. Two important user schemas (geometry and interface-description) have been developed. The use of a commercial relational database system to manage complex VLSI designs has been demonstrated. The performance of an integrated CAD system with INGRES as the heart of the system has been demonstrated by a design example of a programmable memory array chip, MA422.

However, many critical issues have been purposely neglected in this research. Important issues such as maintaining the consistency between various representations of the same object, the functional dependencies of design data, keeping versions of a design object, handling of text-strings, and so forth, have been left for future work.

an
on
nt
to
he
he
he
he
he
al
se
he
AD

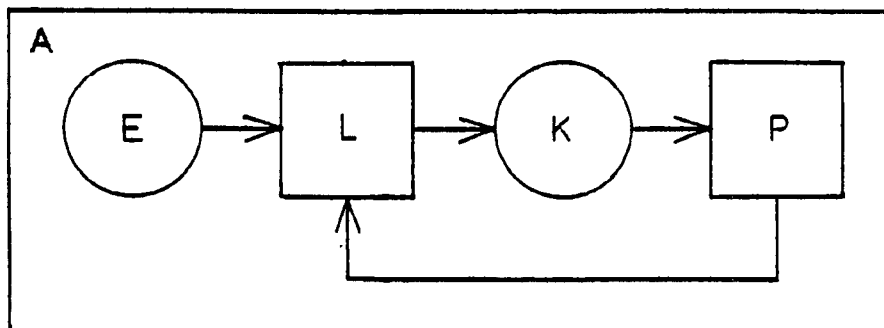


Figure 19. Model for a Learning System

-84-

CONFIDENTIAL

FOO 000127

REFERENCES

1. R. H. Katz, "Managing the Chip Design Database", Computer, Vol. 16, No. 12, Dec. 1983, pp. 26-35.
2. T. Neumann, "On Representing the Design Information in a Common Database", Engineering Design Applications Proceedings of Annual Meeting - Database Week, San Jose, Ca, May 23-26, 1983, pp. 81-87.
3. M. Stonebraker, B. Rubenstein, A. Guttman, "Application of Abstract Data Types and Abstract Indices to CAD Data Bases", Engineering Design Applications Proceedings of Annual Meeting - Database Week, San Jose, Ca, May 23-26, 1983, pp. 107-113.
4. H. Kobayashi and Y. P. Foo, "Programmable Logic for Parallel Convolution" , International Conference on Computer Design ICCD, October 7-11, 1984, Port Chester, New York.
5. H. Kobayashi and R. D. Bonnell, "Arithmetic for a High-Speed Adaptive Learning Network Element," 6th Int. Conf. Computer Arithmetic, Aarhus, Denmark, June 1983, pp. 164-168.
6. H. Kobayashi and T. A. Smith, "SPAN: A Synthesis Program for Adder Networks," 1st Int. Conf. Computers and Applications, Peking, China, June 1984.

7. E. E. Swartzlander, "Merged Arithmetic," IEEE Trans. Comput., vol. C-29, pp. 946-950, Oct. 1980.
8. H. Kobayashi and C. E. Drozd, "Efficient Algorithms for Routing Interchangeable Terminals," International Conference on Computer-Aided Design ICCAD, November 12-15, 1984, Santa Clara, CA.
9. K. Keller, "KIC: A Graphics Editor for Integrated Circuits," Master's Thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, CA. June 1981.
10. C. A. Mead and L. A. Conway, "Introduction to VLSI systems," Addison-Wesley, Reading, Mass., 1980.

APPENDIX

(Symbol PadBlank

.);

DS 0 1 1;

LNM;

B 10600 800 5300 -400;

B 8200 800 5300 -10200;

B 5400 5400 5300 -5300;

LNG;

B 4600 4600 5300 -5300;

DF;

(Symbol PadDriver

.);

DS 1 1 1;

C 0;

LND;

B 10600 2800 5300 -1400;

B 2800 7800 1400 -6700;

LNC;

B 400 200 800 -400;

LNP;

B 200 9200 900 -5600;

B 1700 200 1650 -10100;

B 9000 200 5300 -1000;

LNM;

B 400 600 1300 -2800;

B 400 600 1300 -5300;

B 400 600 1300 -7800;

LNC;

B 200 400 1300 -2800;

B 200 400 1300 -5300;

B 200 400 1300 -7800;

LNM;

B 1300 400 1950 -2800;

B 1300 400 1950 -5300;

B 1300 400 1950 -7800;

LNC;

B 400 200 1800 -400;

LNP;

B 200 7800 1700 -5700;

B 1900 200 2550 -9500;

B 7400 200 5300 -1800;

B 200 600 2400 -10300;

LNM;

B 600 400 2800 -1400;

B 400 1100 2800 -2050;

LNC;

B 400 200 2800 -1400;

B 400 200 2800 -400;

LND;
B 5000 2800 5300 -9200;
LNP;
B 200 1200 3400 -10000;
LNC;
B 400 200 3800 -400;
B 400 200 4100 -10400;
B 400 200 4100 -10000;
B 400 200 4800 -400;
B 400 200 4900 -10400;
B 400 200 4900 -10000;
LNM;
B 600 400 5300 -1400;
LNC;
B 400 200 5300 -1400;
LNM;
B 400 1100 5300 -2050;
LNC;
B 400 200 5700 -10400;
B 400 200 5700 -10000;
B 400 200 5800 -400;
B 400 200 6500 -10400;
B 400 200 6500 -10000;
B 400 200 6800 -400;
LNP;
B 1900 200 8050 -9500;
B 200 1200 7200 -10000;
LNM;
B 600 400 7800 -1400;
LNC;
B 400 200 7800 -1400;
LNM;
B 400 1100 7800 -2050;
LNC;
B 400 200 7800 -400;
LND;
B 2800 7800 9200 -6700;
LNM;
B 1300 400 8650 -7800;
B 1300 400 8650 -5300;
B 1300 400 8650 -2800;
LNP;
B 1700 200 8950 -10100;
B 200 600 8200 -10300;
LNC;
B 400 200 8800 -400;
LNP;
B 200 7800 8900 -5700;
LNM;
B 400 600 9300 -2800;
B 400 600 9300 -5300;
B 400 600 9300 -7800;
LNC;

B 200 400 9300 -2800;
B 200 400 9300 -5300;
B 200 400 9300 -7800;
B 400 200 9800 -400;
LNP;
B 200 9200 9700 -5600;
DF;

(Symbol padgnd.k);
(Creation Date: Wed Aug 1 21:32:21 1984
.);

DS 2 1 1;
L NM;
B 10000 800 5000 400;
B 10000 1600 5000 9200;
B 8200 2000 5000 8800;
B 800 1800 2700 8900;
B 5400 5400 5000 5300;
B 800 1800 7300 8900;
L NG;
B 4600 4600 5000 5300;
DF;

(Symbol PadIn
.);
DS 3 1 1;
C 0;
LND;
B 7700 1800 5050 -9300;
LNC;
B 400 200 1500 -10000;
LNP;
B 200 700 2000 -9250;
B 7500 200 5650 -8900;
B 7500 200 5650 -9600;
LNC;
B 400 200 2500 -10000;
LNM;
B 400 1700 2800 -2050;
LND;
B 400 800 2800 -1400;
LNC;
B 200 400 2800 -1500;
LND;
B 6700 200 6250 -1100;
LNC;
B 400 200 3500 -10000;
B 400 200 4500 -10000;
B 400 200 5500 -10000;
B 400 200 6500 -10000;
B 400 200 7500 -10000;
B 400 200 8500 -10000;
LND;

B 800 1300 9200 -9250;
 LNP;
 B 400 400 9200 -10200;
 LNC;
 B 200 200 9200 -10200;
 LNP;
 B 200 500 9300 -8650;
 B 200 800 9300 -10000;
 LND;
 B 200 9600 9600 -5800;
 DF;

(Symbol PadOut8
 .);
 DS 4 1 1;
 C 5;
 LND;
 B 1200 700 5600 -14150;
 B 800 600 8000 -14100;
 LNP;
 B 900 200 8150 -14100;
 DF;

(Symbol PadOut4
 .);
 DS 5 1 1;
 C 1;
 LND;
 B 400 2700 900 -11950;
 B 1000 400 1400 -13100;
 B 800 1200 1400 -11500;
 LNI;
 B 600 1600 1400 -11500;
 B 600 800 1400 -13100;
 LNM;
 B 400 600 1400 -13700;
 LNP;
 B 400 300 1400 -13550;
 LND;
 B 600 400 1500 -13800;
 LNP;
 B 200 2800 1400 -12100;
 LNC;
 B 200 400 1400 -13700;
 LNP;
 B 1400 200 2100 -12300;
 LND;
 B 200 1500 1800 -13750;
 B 800 400 2100 -11700;
 B 1200 200 2400 -14400;
 LNM;
 B 600 900 2200 -11450;
 LNP;

B 600 400 2200 -11200;
LNC;
B 400 200 2200 -11200;
B 400 200 2200 -11700;
LNM;
B 400 1500 2300 -12250;
B 1400 400 2800 -13100;
LNP;
B 200 900 2400 -10850;
B 200 1400 2700 -13400;
B 6600 200 5900 -12700;
B 2100 200 3650 -14100;
B 5300 200 5250 -11900;
B 200 400 2700 -12100;
LND;
B 2400 1600 4100 -12300;
B 1600 700 3700 -14150;
B 600 400 3200 -13100;
LNM;
B 1300 400 3550 -11200;
LNP;
B 600 400 3200 -11200;
LNC;
B 400 200 3200 -13100;
B 400 200 3200 -11200;
LNP;
B 200 900 3400 -10850;
LND;
B 600 500 3900 -11250;
LNC;
B 400 200 3900 -11200;
LNM;
B 2600 400 5300 -12300;
B 2600 400 5300 -13700;
LND;
B 2600 400 5300 -13700;
LNC;
B 400 200 4300 -12300;
B 400 200 4300 -13700;
LNM;
B 800 3400 5300 -12100;
LNC;
B 400 200 5300 -12300;
B 400 200 5300 -13700;
LNP;
B 200 500 5300 -14250;
LND;
B 2400 1600 6500 -12300;
LNP;
B 2600 200 6600 -14100;
LNC;
B 400 200 6300 -12300;
LND;

B 1600 700 6900 -14150;
LNC;
B 400 200 6300 -13700;
LND;
B 600 500 6700 -11250;
LNM;
B 1300 400 7050 -11200;
LNC;
B 400 200 6700 -11200;
LNP;
B 200 900 7200 -10850;
LND;
B 600 400 7400 -13100;
LNM;
B 1400 400 7800 -13100;
LNP;
B 600 400 7400 -11200;
LNC;
B 400 200 7400 -13100;
B 400 200 7400 -11200;
LND;
B 1200 200 8200 -11600;
B 1200 200 8200 -14400;
LNP;
B 200 900 8200 -10850;
LNM;
B 400 2000 8300 -12000;
LNP;
B 600 400 8400 -11200;
LNM;
B 600 400 8400 -11200;
LNC;
B 400 200 8400 -11200;
LND;
B 1000 400 9200 -13100;
B 200 1500 8800 -13750;
B 800 1200 9200 -11500;
B 600 400 9100 -13800;
LNI;
B 600 1600 9200 -11500;
B 600 800 9200 -13100;
LNM;
B 400 600 9200 -13700;
LNP;
B 400 300 9200 -13550;
B 200 2800 9200 -12100;
LNC;
B 200 400 9200 -13700;
LND;
B 400 2700 9700 -11950;
DF;

(Symbol PadTriState

.);
DS 6 1 1;
C 1;
LND;
B 200 1900 300 -13750;
LNM;
B 600 400 500 -13700;
LND;
B 400 400 400 -13700;
LNI;
B 600 1000 600 -13100;
LNC;
B 400 200 500 -13700;
LND;
B 600 2400 600 -15700;
B 400 600 600 -13100;
LNP;
B 200 1000 600 -13100;
B 300 400 650 -13700;
B 200 2700 600 -15650;
B 300 200 850 -13800;
LND;
B 400 2800 1000 -12000;
B 6900 400 4250 -16800;
LNP;
B 200 700 1000 -14050;
B 1400 200 1600 -14300;
LND;
B 800 400 1400 -13200;
B 800 1200 1400 -12000;
LNI;
B 600 800 1400 -13200;
B 600 1600 1400 -12000;
LNP;
B 200 2500 1400 -12450;
LNM;
B 400 2700 1500 -14850;
LNP;
B 400 300 1500 -13650;
LND;
B 600 400 1600 -13900;
LNP;
B 1300 200 2050 -12800;
LNC;
B 200 400 1500 -13800;
LND;
B 800 400 2100 -12400;
LNM;
B 1800 400 2600 -16000;
LND;
B 200 1100 1900 -13550;
LNP;
B 600 400 2200 -11800;

LNM;
B 600 1000 2200 -12100;
LNC;
B 400 200 2200 -11800;
B 400 200 2200 -12400;
LNM;
B 400 2600 2300 -13300;
LNP;
B 200 2500 2300 -15150;
B 5700 200 5050 -16400;
B 5700 200 5150 -14000;
B 200 1100 2400 -11150;
LNM;
B 1100 400 2950 -14400;
LNP;
B 200 800 2700 -15200;
B 5900 200 5550 -14800;
B 200 900 2700 -13550;
B 2100 200 3650 -15600;
B 200 600 2700 -12600;
B 5200 200 5300 -12400;
B 5200 200 5300 -13200;
LNM;
B 600 1400 3200 -12300;
LND;
B 1600 1700 3700 -16150;
B 700 2800 3250 -14000;
LNP;
B 600 400 3200 -11800;
LNM;
B 4800 400 5300 -16800;
LNC;
B 400 200 3200 -12800;
B 400 200 3200 -14400;
B 400 200 3200 -16000;
B 400 200 3200 -16800;
B 400 200 3200 -11800;
LNP;
B 200 1100 3400 -11150;
LND;
B 3400 3600 5300 -13600;
LNM;
B 2600 400 5300 -12000;
B 2600 400 5300 -13600;
B 2600 400 5300 -15200;
LNC;
B 400 200 4200 -16800;
B 400 200 4300 -13600;
B 400 200 4300 -12000;
B 400 200 4300 -15200;
LNM;
B 800 6200 5300 -13700;
LNC;

B 400 200 5300 -16800;
B 400 200 5300 -13600;
B 400 200 5300 -12000;
B 400 200 5300 -15200;
LNP;
B 2400 200 7100 -15600;
LNC;
B 400 200 6300 -13600;
B 400 200 6300 -12000;
B 400 200 6300 -15200;
LND;
B 1600 1700 6900 -16150;
LNC;
B 400 200 6400 -16800;
LND;
B 700 3300 7350 -13750;
LNM;
B 600 1400 7400 -12300;
B 1400 400 7800 -14400;
B 1900 400 8050 -16000;
LNP;
B 200 1100 7200 -11150;
B 600 400 7400 -11800;
LNC;
B 400 200 7400 -12800;
B 400 200 7400 -14400;
B 400 200 7400 -16000;
B 400 200 7400 -16800;
B 400 200 7400 -11800;
LND;
B 600 700 7700 -10950;
LNM;
B 600 400 7700 -11100;
LNC;
B 400 200 7700 -11100;
LND;
B 600 400 7900 -12800;
LNP;
B 200 900 7900 -13550;
LNM;
B 2300 300 9050 -11150;
LND;
B 300 700 8150 -12650;
B 900 400 8450 -12400;
LNM;
B 400 2700 8300 -13250;
LNP;
B 200 1100 8200 -11150;
B 600 400 8400 -11800;
LNM;
B 600 400 8400 -11800;
B 600 400 8400 -11800;
LNP;

B 200 1500 8200 -16250;
LNC;
B 400 200 8400 -11800;
LNP;
B 200 2200 8500 -13800;
B 800 200 8800 -12800;
LND;
B 200 1100 8800 -13550;
B 600 400 9100 -13900;
B 800 400 9200 -13200;
B 800 1200 9200 -12000;
LNI;
B 600 800 9200 -13200;
B 600 1600 9200 -12000;
LNM;
B 400 2700 9200 -14850;
LNP;
B 400 300 9200 -13650;
LNC;
B 200 400 9200 -13800;
LNP;
B 200 2500 9200 -12450;
LND;
B 400 2800 9700 -12000;
LNM;
B 300 6000 10250 -14000;
DF;

(Symbol padvdd.k);

(Creation Date: Wed Aug 1 23:52:21 1984
.);

DS 7 1 1;

L NM;

B 800 10000 400 5000;
B 2000 5400 1600 5000;
B 5400 5400 5300 5000;
B 1800 800 1900 2700;
B 1800 800 1900 7300;

L NG;

B 4600 4600 5300 5000;
DF;

(Symbol placonn.k);

DS 8 1 1;

L ND;

B 500 400 1050 1000;

L NP;

B 800 200 900 600;
B 800 200 900 1400;
B 400 400 300 1300;
B 400 400 300 500;

L NM;

B 500 400 250 500;

B 500 400 250 1300;
B 400 1600 1000 800;
L NC;
B 200 200 300 500;
B 200 200 300 1300;
DF;

(Symbol plagnd.k);
(Creation Date: Wed May 16 22:52:21 1984
.);

DS 9 1 1;
L ND;
B 400 800 600 400;
L NP;
B 200 800 1000 400;
B 200 800 200 400;
L NM;
B 1600 400 800 500;
L NC;
B 200 200 600 500;
DF;

(Symbol plain.k);
DS 10 1 1;
L NI;
B 600 600 1200 1800;
B 600 600 1200 2600;
L ND;
B 300 200 250 3400;
B 600 800 600 3500;
B 400 600 1100 3100;
B 200 400 1200 2600;
B 400 400 1200 2200;
B 200 400 1200 1800;
B 400 600 1100 1300;
B 500 800 650 1200;
B 300 2900 250 1950;
B 400 400 200 300;
B 400 500 1000 250;
L NP;
B 700 200 1250 1800;
B 200 300 1500 1550;
B 400 400 1400 1200;
B 200 1000 1400 500;
B 400 400 1400 3200;
B 200 300 1500 2850;
B 900 200 1150 2600;
B 200 2700 600 1350;
B 200 1200 600 3500;
L NM;
B 600 400 1200 3200;
B 1600 400 800 2200;
B 600 400 1200 1200;

- 97 -

B 1600 400 800 300;
 L NC;
 B 400 200 1200 3200;
 B 200 200 1200 2200;
 B 400 200 1200 1200;
 B 200 200 200 300;
 B 200 200 1000 300;
 DF;

(Symbol plaout.k);
 DS 11 1 1;
 L NI;
 B 600 600 1100 900;
 B 600 600 800 1500;
 L ND;
 B 200 2100 1400 1550;
 B 400 400 1300 300;
 B 400 200 1100 900;
 B 600 500 600 1050;
 B 200 400 800 1500;
 B 500 400 650 1900;
 B 200 1500 300 2450;
 B 600 800 1100 2800;
 B 600 800 600 3400;
 B 400 400 1000 3800;
 L NP;
 B 200 1100 1100 2750;
 B 500 200 1250 3400;
 B 200 800 1400 3900;
 B 200 1500 600 3550;
 B 400 400 1300 4500;
 B 400 400 500 4500;
 B 300 400 1050 300;
 B 200 1200 1100 600;
 B 200 700 300 350;
 B 200 1200 100 1100;
 B 400 400 400 1600;
 B 500 200 850 1500;
 L NM;
 B 400 700 400 1750;
 B 600 400 1200 300;
 B 400 600 1300 4600;
 B 400 600 500 4600;
 B 1600 400 800 3800;
 B 1600 400 800 1000;
 L NC;
 B 200 500 400 1750;
 B 400 200 1200 300;
 B 200 200 500 1000;
 B 200 200 1000 3800;
 B 200 200 1300 4500;
 B 200 200 500 4500;
 DF;

```

(Symbol cellpair.k);
DS 12 1 1;
L ND;
B 400 1600 600 800;
L NP;
B 200 1600 200 800;
B 200 1600 1000 800;
L NM;
B 1600 400 800 1300;
B 1600 400 800 500;
DF;

```

```

(Symbol pullup2.k);
DS 13 1 1;
L NI;
B 900 600 950 500;
B 900 600 950 1300;
L ND;
B 400 400 200 500;
B 400 400 200 1300;
B 700 200 750 500;
B 700 200 750 1300;
B 400 400 1300 500;
B 400 400 1300 1300;
L NP;
B 500 600 950 500;
B 500 600 950 1300;
L NM;
B 400 1600 200 800;
B 600 400 1200 1300;
B 600 400 1200 500;
L NC;
B 400 200 1200 500;
B 400 200 1200 1300;
B 200 200 200 500;
B 200 200 200 1300;
DF;

```

```

(Symbol pmabl.k);
DS 14 1 1;
C 12 MY R 0 1 T 14000 -1200;
C 12 MY R 0 1 T 12400 -1200;
C 12 MY R 0 1 T 10800 -1200;
C 12 MY R 0 1 T 9200 -1200;
C 12 T 6300 -1600;
C 12 T 4700 -1600;
C 12 T 3100 -1600;
C 12 T 1500 -1600;
C 8 T 7900 -1600;
C 13 T 0 -1600;
C 12 MY R 0 1 T 14000 5200;
C 12 MY R 0 1 T 14000 3600;
C 12 MY R 0 1 T 12400 5200;

```

C 12 MY R 0 1 T 12400 3600;
C 12 MY R 0 1 T 10800 5200;
C 12 MY R 0 1 T 10800 3600;
C 12 MY R 0 1 T 14000 2000;
C 12 MY R 0 1 T 14000 400;
C 12 MY R 0 1 T 12400 2000;
C 12 MY R 0 1 T 12400 400;
C 12 MY R 0 1 T 10800 2000;
C 12 MY R 0 1 T 10800 400;
C 12 MY R 0 1 T 9200 400;
C 12 MY R 0 1 T 9200 2000;
C 12 MY R 0 1 T 9200 3600;
C 12 MY R 0 1 T 9200 5200;
C 12 T 6300 0;
C 12 T 6300 1600;
C 12 T 6300 3200;
C 12 T 6300 4800;
C 12 T 4700 4800;
C 12 T 4700 3200;
C 12 T 4700 1600;
C 12 T 4700 0;
C 12 T 3100 4800;
C 12 T 3100 3200;
C 12 T 3100 1600;
C 12 T 3100 0;
C 12 T 1500 4800;
C 12 T 1500 3200;
C 12 T 1500 1600;
C 12 T 1500 0;
C 8 T 7900 4800;
C 8 T 7900 3200;
C 8 T 7900 1600;
C 8 T 7900 0;
C 13 R 0 -1 T 14000 8300;
C 13 R 0 -1 T 12400 8300;
C 13 R 0 -1 T 10800 8300;
C 13 R 0 -1 T 9200 8300;
C 13 T 0 4800;
C 13 T 0 3200;
C 13 T 0 1600;
C 13 T 0 0;
C 10 T 5900 6400;
C 10 T 4300 6400;
C 10 T 2700 6400;
C 10 T 1100 6400;
C 12 MY R 0 1 T 14000 -2800;
C 12 MY R 0 1 T 12400 -2800;
C 12 MY R 0 1 T 10800 -2800;
C 12 MY R 0 1 T 14000 -4400;
C 12 MY R 0 1 T 12400 -4400;
C 12 MY R 0 1 T 10800 -4400;
C 12 MY R 0 1 T 14000 -6000;
C 12 MY R 0 1 T 12400 -6000;

- 100 -

C 12 MY R 0 1 T 10800 -6000;
C 12 MY R 0 1 T 9200 -6000;
C 12 MY R 0 1 T 9200 -4400;
C 12 MY R 0 1 T 9200 -2800;
C 12 T 6300 -6400;
C 12 T 4700 -6400;
C 12 T 3100 -6400;
C 12 T 6300 -4800;
C 12 T 4700 -4800;
C 12 T 3100 -4800;
C 12 T 6300 -3200;
C 12 T 4700 -3200;
C 12 T 3100 -3200;
C 12 T 1500 -3200;
C 12 T 1500 -4800;
C 12 T 1500 -6400;
C 8 T 7900 -3200;
C 8 T 7900 -4800;
C 8 T 7900 -6400;
C 13 T 0 -6400;
C 13 T 0 -3200;
C 13 T 0 -4800;
C 11 T 14000 -10900;
C 11 T 12400 -10900;
C 11 T 10800 -10900;
C 11 T 9200 -10900;
L ND;
B 200 400 12900 -200;
B 200 400 12100 -200;
B 200 400 11300 -200;
B 200 400 10500 -200;
B 200 400 12900 3800;
B 400 400 12900 3400;
B 200 400 14500 3800;
B 200 400 12100 3800;
B 200 400 10500 3800;
B 400 400 10500 3400;
B 400 400 14500 3400;
B 200 400 14500 3000;
B 400 400 13700 3400;
B 200 400 13700 3000;
B 200 400 12100 3000;
B 400 400 12100 3400;
B 400 400 9700 3400;
B 200 400 9700 3000;
B 200 400 12900 2200;
B 200 400 12100 2200;
B 200 400 11300 2200;
B 400 400 11300 1800;
B 200 400 10500 2200;
B 400 400 14500 1800;
B 200 400 14500 1400;
B 400 400 12900 1800;

- 101 -

B 200 400 12900 1400;
B 400 400 12100 1800;
B 200 400 12100 1400;
B 400 400 10500 1800;
B 200 400 10500 1400;
B 400 400 13700 200;
B 200 400 13700 600;
B 200 400 12900 600;
B 200 400 12100 600;
B 200 400 11300 600;
B 400 400 12900 200;
B 400 400 12100 200;
B 400 400 11300 200;
B 400 400 10500 200;
B 400 400 12900 5000;
B 200 400 12900 4600;
B 200 400 12100 4600;
B 400 400 11300 5000;
B 200 400 11300 4600;
B 400 400 10500 5000;
B 200 400 10500 4600;
B 200 400 14500 5400;
B 400 400 14500 5000;
B 200 400 13700 5400;
B 400 400 13700 5000;
B 400 400 12100 5000;
B 200 400 12100 5400;
B 200 800 13700 6400;
B 200 800 12100 6400;
B 200 800 11300 6400;
B 400 400 9700 5000;
B 200 400 9700 5400;
B 200 800 9700 6400;
B 400 200 1700 6100;
B 400 200 1700 5300;
B 400 200 1700 4500;
B 400 400 2900 6100;
B 400 200 3300 6100;
B 400 400 2900 5300;
B 400 200 3300 5300;
B 400 400 2900 4500;
B 400 200 3300 4500;
B 400 200 1700 3700;
B 400 400 2900 3700;
B 400 200 3300 3700;
B 400 200 1700 2900;
B 400 200 1700 2100;
B 400 200 4100 2900;
B 400 400 4500 2900;
B 400 200 4100 2100;
B 400 400 4500 2100;
B 400 200 1700 1300;
B 400 200 1700 500;

B 400 200 4100 1300;
B 400 400 4500 1300;
B 400 200 4100 500;
B 400 400 4500 500;
B 400 400 4500 6100;
B 400 200 4900 6100;
B 400 400 6100 6100;
B 400 200 6500 6100;
B 400 400 4500 5300;
B 400 200 4900 5300;
B 400 200 5700 4500;
B 400 400 6100 4500;
B 400 200 5700 3700;
B 400 400 6100 3700;
B 400 200 4900 2900;
B 400 200 4900 2100;
B 400 200 5700 1300;
B 400 400 6100 1300;
B 400 200 5700 500;
B 400 400 6100 500;
B 400 200 7300 500;
B 400 400 7700 500;
B 400 200 6500 1300;
B 400 200 7300 2100;
B 400 400 7700 2100;
B 400 400 6100 2900;
B 400 200 6500 2900;
B 400 200 7300 3700;
B 400 400 7700 3700;
B 400 200 6500 4500;
B 400 200 7300 5300;
B 400 400 7700 5300;
B 200 400 10500 -1000;
B 400 400 10500 -1400;
B 200 400 12100 -1000;
B 200 400 12900 -1000;
B 200 400 14500 -1000;
B 400 400 14500 -1400;
B 400 400 13700 -1400;
B 200 400 13700 -1800;
B 400 400 12900 -1400;
B 200 400 12900 -1800;
B 400 400 12100 -1400;
B 200 400 12100 -1800;
B 400 400 11300 -1400;
B 200 400 11300 -1800;
B 200 400 12100 -2600;
B 200 400 12900 -2600;
B 400 400 13700 -3000;
B 200 400 13700 -2600;
B 200 400 14500 -2600;
B 400 400 14500 -3000;
B 200 400 14500 -3400;

B 400 400 12900 -3000;
B 200 400 12900 -3400;
B 400 400 12100 -3000;
B 200 400 12100 -3400;
B 400 400 10500 -3000;
B 200 400 10500 -3400;
B 400 400 11300 -4600;
B 200 400 11300 -4200;
B 200 400 12100 -4200;
B 200 400 12900 -4200;
B 200 400 13700 -4200;
B 400 400 14500 -4600;
B 200 400 14500 -5000;
B 400 400 13700 -4600;
B 200 400 13700 -5000;
B 400 400 12900 -4600;
B 200 400 12900 -5000;
B 400 400 12100 -4600;
B 200 400 12100 -5000;
B 400 400 2100 -6600;
B 400 400 3700 -6600;
B 400 400 5300 -6600;
B 400 400 6900 -6600;
B 400 200 2500 -300;
B 400 400 2900 -300;
B 400 200 3300 -300;
B 400 200 2500 -1100;
B 400 400 2900 -1100;
B 400 200 3300 -1100;
B 400 200 2500 -1900;
B 400 400 2900 -1900;
B 400 200 3300 -1900;
B 400 200 2500 -2700;
B 400 400 2900 -2700;
B 400 200 3300 -2700;
B 400 200 2500 -3500;
B 400 400 2900 -3500;
B 400 200 4100 -3500;
B 400 400 4500 -3500;
B 400 200 4100 -4300;
B 400 400 4500 -4300;
B 400 200 4100 -5100;
B 400 400 4500 -5100;
B 400 200 2500 -5100;
B 400 400 2900 -5100;
B 400 200 2500 -4300;
B 400 400 2900 -4300;
B 400 200 2500 -5900;
B 400 400 2900 -5900;
B 400 200 4100 -5900;
B 400 400 4500 -5900;
B 400 400 4500 -300;
B 400 200 4900 -300;

B 400 400 4500 -1100;
B 400 200 4900 -1100;
B 400 200 5700 -1900;
B 400 400 6100 -1900;
B 400 200 5700 -2700;
B 400 400 6100 -2700;
B 400 200 4900 -3500;
B 400 200 4900 -4300;
B 400 200 5700 -5100;
B 400 400 6100 -5100;
B 400 200 5700 -5900;
B 400 400 6100 -5900;
B 400 200 7300 -5900;
B 400 400 7700 -5900;
B 400 200 6500 -5100;
B 400 200 7300 -4300;
B 400 400 7700 -4300;
B 400 400 6100 -3500;
B 400 200 6500 -3500;
B 400 200 6500 -1900;
B 400 200 7300 -2700;
B 400 400 7700 -2700;
B 400 200 7300 -1100;
B 400 400 7700 -1100;
B 400 400 6100 -300;
B 400 200 6500 -300;
L NM;
B 15600 400 7800 15600;
B 15600 400 7800 14900;
B 15600 400 7800 14200;
B 15600 400 7800 13500;
B 15600 400 7800 12800;
B 15600 400 7800 12100;
B 15600 400 7800 11400;
B 2600 400 14300 10700;
B 400 2400 200 7600;
B 700 400 750 8600;
B 1300 400 8150 8600;
B 400 900 9000 8350;
B 1200 400 8100 6700;
B 400 500 8900 6650;
B 7500 400 3750 10700;
B 5500 400 10250 10700;
B 500 900 8950 -6850;
B 6800 400 5300 -6600;
B 400 1100 200 -6950;
B 8000 400 4400 -7300;
B 400 2600 8200 -8800;
B 800 400 8800 -9900;
L NC;
B 200 200 12900 3400;
B 200 200 10500 3400;
B 200 200 14500 3400;

B 200 200 13700 3400;
B 200 200 12100 3400;
B 200 200 9700 3400;
B 200 200 11300 1800;
B 200 200 10500 1800;
B 200 200 12100 1800;
B 200 200 12900 1800;
B 200 200 14500 1800;
B 200 200 13700 200;
B 200 200 12900 200;
B 200 200 12100 200;
B 200 200 11300 200;
B 200 200 10500 200;
B 200 200 12900 5000;
B 200 200 11300 5000;
B 200 200 10500 5000;
B 200 200 14500 5000;
B 200 200 13700 5000;
B 200 200 12100 5000;
B 200 200 9700 5000;
B 200 200 2900 6100;
B 200 200 2900 5300;
B 200 200 2900 4500;
B 200 200 2900 3700;
B 200 200 4500 2900;
B 200 200 4500 2100;
B 200 200 4500 1300;
B 200 200 4500 500;
B 200 200 4500 6100;
B 200 200 4500 5300;
B 200 200 6100 6100;
B 200 200 6100 4500;
B 200 200 6100 3700;
B 200 200 6100 500;
B 200 200 6100 1300;
B 200 200 7700 500;
B 200 200 7700 2100;
B 200 200 6100 2900;
B 200 200 7700 3700;
B 200 200 7700 5300;
B 200 200 10500 -1400;
B 200 200 14500 -1400;
B 200 200 11300 -1400;
B 200 200 12100 -1400;
B 200 200 12900 -1400;
B 200 200 13700 -1400;
B 200 200 13700 -3000;
B 200 200 12100 -3000;
B 200 200 12900 -3000;
B 200 200 14500 -3000;
B 200 200 10500 -3000;
B 200 200 11300 -4600;
B 200 200 12100 -4600;

B 200 200 12900 -4600;
 B 200 200 13700 -4600;
 B 200 200 14500 -4600;
 B 200 200 2100 -6600;
 B 200 200 3700 -6600;
 B 200 200 5300 -6600;
 B 200 200 6900 -6600;
 B 200 200 2900 -300;
 B 200 200 2900 -1100;
 B 200 200 2900 -1900;
 B 200 200 2900 -2700;
 B 200 200 2900 -3500;
 B 200 200 4500 -3500;
 B 200 200 2900 -4300;
 B 200 200 4500 -4300;
 B 200 200 2900 -5100;
 B 200 200 4500 -5100;
 B 200 200 2900 -5900;
 B 200 200 4500 -5900;
 B 200 200 6100 -2700;
 B 200 200 6100 -1900;
 B 200 200 4500 -1100;
 B 200 200 6100 -5100;
 B 200 200 6100 -5900;
 B 200 200 7700 -5900;
 B 200 200 7700 -4300;
 B 200 200 6100 -3500;
 B 200 200 7700 -2700;
 B 200 200 7700 -1100;
 B 200 200 6100 -300;
 B 200 200 4500 -300;
 DF;

(Symbol n4p2q2.k);
 DS 15 l l;
 C 4 MY T -1500 -31000;
 C 3 MY T -10600 18500;
 C 14 T 16500 0;
 C 14 T 0 0;
 C 3 MY T -31800 18500;
 C 3 MY T -21200 18500;
 C 3 MY T 42400 18500;
 C 3 MY T 31800 18500;
 C 3 MY T 21200 18500;
 C 3 MY T 10600 18500;
 C 3 MY T 0 18500;
 C 7 MX T 43500 400;
 C 4 MY T -34500 -31000;
 C 4 MY T -45500 -31000;
 C 4 MY T -56500 -31000;
 C 4 MY T -67500 -31000;
 C 4 MY T -12500 -31000;
 C 4 MY T -23500 -31000;

C 4 MY T 75500 -31000;
C 4 MY T 64500 -31000;
C 4 MY T 53500 -31000;
C 4 MY T 42500 -31000;
C 4 MY T 31500 -31000;
C 4 MY T 20500 -31000;
C 4 MY T 9500 -31000;
C 2 MX R 0 -1 T 33500 -10500;
L ND;
B 2400 200 -1000 13500;
B 1800 200 -700 14200;
B 1300 200 -450 14900;
B 200 1500 -22200 18750;
B 12500 200 -16050 17900;
B 200 700 -11600 19150;
B 1900 200 -10750 18700;
B 7600 200 -6000 17900;
B 8200 200 -5700 18700;
B 200 800 -1500 18400;
B 200 4500 -2100 15750;
B 200 3700 -1500 16150;
B 200 4500 -1000 17250;
B 200 1300 9600 18850;
B 9200 200 5100 18100;
B 200 2400 400 17000;
B 200 3100 52000 17950;
B 9900 200 47150 16300;
B 200 2700 41400 18150;
B 7900 200 37550 16700;
B 8000 200 38200 16300;
B 200 2600 34100 15100;
B 200 3000 33500 15300;
B 200 1200 32900 14400;
B 200 3000 34100 12300;
B 200 2300 33500 12650;
B 2400 200 33000 10700;
B 1800 200 32700 11400;
B 200 1600 32900 13000;
B 1200 200 32400 12100;
B 200 1600 32900 15800;
B 2100 200 31950 16700;
B 200 2900 30800 18050;
B 200 3100 32300 14450;
B 600 200 32100 12800;
B 7600 200 28600 16100;
B 4500 200 22550 16100;
B 200 3500 20200 17750;
B 400 400 31600 12800;
B 400 400 31600 12100;
B 400 400 31600 10700;
B 400 400 31600 11400;
B 400 400 400 12800;
B 400 400 400 13500;

- 108 -

B 400 400 400 14200;
B 400 400 400 14900;
B 400 400 400 15600;
B 400 1400 18600 -7500;
L NP;
B 11900 200 -1350 -13700;
B 11900 200 -1350 -14100;
B 11900 200 -1350 -14500;
B 11900 200 -1350 -14900;
B 11900 200 -1350 -15300;
B 11700 200 -1250 -15700;
B 400 400 23000 10700;
B 400 400 21400 11400;
B 200 700 21400 10850;
B 200 2800 19800 11900;
B 400 400 19800 13500;
B 400 400 18200 14200;
B 200 1000 18200 13500;
B 200 2500 18200 11750;
B 400 400 6500 12100;
B 200 1400 6500 11200;
B 400 400 4900 12800;
B 200 2100 4900 11550;
B 400 400 3300 14900;
B 200 4200 3300 12600;
B 400 400 1700 15600;
B 200 4900 1700 12950;
B 200 2700 -62200 -15150;
B 11000 200 -56800 -13700;
B 11000 200 -45800 -13700;
B 200 2300 -51200 -15350;
B 11000 200 -45800 -14100;
B 11000 200 -34800 -13700;
B 11000 200 -34800 -14100;
B 200 1900 -40200 -15550;
B 11000 200 -34800 -14500;
B 11000 200 -23800 -13700;
B 11000 200 -23800 -14100;
B 11000 200 -23800 -14500;
B 200 1500 -29200 -15750;
B 11000 200 -23800 -14900;
B 11000 200 -12800 -13700;
B 11000 200 -12800 -14100;
B 11000 200 -12800 -14500;
B 11000 200 -12800 -14900;
B 200 1100 -18200 -15950;
B 11000 200 -12800 -15300;
B 200 900 -7200 -16050;
B 200 2700 9500 -12250;
B 5000 200 7100 -13700;
B 5800 200 7500 -14100;
B 200 400 11100 -14200;
B 6600 200 7900 -14500;

B 200 800 11900 -14400;
B 7400 200 8300 -14900;
B 200 1200 12700 -14600;
B 8200 200 8700 -15300;
B 200 1200 13500 -15000;
B 9000 200 9100 -15700;
B 200 3100 10300 -12450;
B 200 3100 11100 -12450;
B 200 3100 11900 -12450;
B 200 3100 12700 -12450;
B 200 3500 13500 -12650;
B 200 300 3800 -16350;
B 2700 200 5050 -16100;
B 7800 200 10300 -16100;
B 200 1800 14300 -15300;
B 200 3500 14300 -12650;
B 200 2700 80800 -15150;
B 4300 200 78750 -13700;
B 12000 200 70600 -13700;
B 13800 200 57700 -13700;
B 11900 200 44850 -13700;
B 8100 200 34950 -13700;
B 200 2900 30800 -12350;
B 200 2100 69800 -15450;
B 7100 200 66350 -14300;
B 12000 200 56800 -14300;
B 12800 200 44400 -14300;
B 7900 200 34050 -14300;
B 200 3500 30000 -12650;
B 200 1500 58800 -15750;
B 8100 200 54850 -14900;
B 12800 200 44400 -14900;
B 8700 200 33650 -14900;
B 200 4100 29200 -12950;
B 200 900 47800 -16050;
B 2800 200 46500 -15500;
B 8800 200 40700 -15500;
B 7800 200 32400 -15500;
B 200 4700 28400 -13250;
B 200 300 36800 -16350;
B 3700 200 35050 -16100;
B 5500 200 30450 -16100;
B 200 5300 27600 -13550;
B 6100 200 17950 -15500;
B 200 1100 14800 -15950;
B 5100 200 23550 -15500;
B 200 4500 26000 -13150;
B 1000 200 26400 -16100;
B 200 500 25800 -16250;
B 200 5100 26800 -13450;
L NM;
B 10000 1000 4000 -11500;
B 1000 4000 -35500 -400;

B 2400 800 0 28700;
B 800 4800 90400 -800;
B 500 11600 32750 -4300;
B 1000 4800 -35500 4000;
B 1000 4800 -35500 8800;
B 1000 4800 -35500 13600;
B 1000 2600 -35500 17300;
B 1000 4800 -35500 21000;
B 1000 4900 -35500 25850;
B 5400 800 -33300 28700;
B 2400 800 -10600 28700;
B 2400 800 -21200 28700;
B 2400 800 42400 28700;
B 2400 800 31800 28700;
B 2400 800 21200 28700;
B 2400 800 10600 28700;
B 16800 800 81600 1200;
B 17600 800 64400 1200;
B 12100 800 49550 1200;
B 9700 500 37850 1250;
B 3000 800 36500 10000;
B 800 2100 34600 9350;
B 4700 800 40350 10000;
B 2900 400 33550 8100;
B 900 400 16050 10700;
B 900 400 16050 11400;
B 900 400 16050 12100;
B 900 400 16050 12800;
B 900 400 16050 13500;
B 900 400 16050 14200;
B 900 400 16050 14900;
B 900 400 16050 15600;
B 900 400 16050 8100;
B 11200 1000 -6600 -11500;
B 12800 1000 -18600 -11500;
B 10000 1000 -30000 -11500;
B 1000 4600 -35500 -9700;
B 1000 5000 -35500 -4900;
B 2800 800 -56700 -20800;
B 400 800 -56700 -30600;
B 2800 800 -45700 -20800;
B 400 800 -45700 -30600;
B 2800 800 -34700 -20800;
B 400 800 -34700 -30600;
B 400 800 -23700 -30600;
B 2800 800 -23700 -20800;
B 2800 800 -12700 -20800;
B 400 800 -12700 -30600;
B 400 800 -1700 -30600;
B 2800 800 -1700 -20800;
B 4600 800 20300 -11600;
B 9000 800 13500 -11600;
B 800 6800 90400 -6600;

- 111 -

B 800 7300 90400 -13650;
B 800 5900 90400 -20250;
B 800 7000 90400 -26700;
B 4800 800 88400 -30600;
B 800 3300 87200 -18850;
B 2800 800 86200 -20800;
B 800 6700 87200 -13850;
B 21600 900 76800 -10050;
B 14000 900 59000 -10050;
B 8500 900 47750 -10050;
B 1000 600 38500 -10800;
B 6500 400 35750 -11300;
B 10400 400 27300 -11300;
B 400 800 22300 -10700;
B 400 1400 22300 -9600;
B 2700 400 21150 -8700;
B 3700 400 17950 -8700;
B 400 1600 15900 -8100;
B 500 400 15850 -7100;
B 2800 800 9300 -20800;
B 400 800 9300 -30600;
B 2800 800 20300 -20800;
B 400 800 20300 -30600;
B 400 800 31300 -30600;
B 2800 800 31300 -20800;
B 2800 800 42300 -20800;
B 400 800 42300 -30600;
B 400 800 53300 -30600;
B 2800 800 53300 -20800;
B 2800 800 64300 -20800;
B 400 800 64300 -30600;
B 400 800 75300 -30600;
B 2800 800 75300 -20800;
B 400 400 32300 -9900;
B 3900 900 39950 -10050;
L NC;
B 200 200 31600 10700;
B 200 200 31600 11400;
B 200 200 31600 12100;
B 200 200 31600 12800;
B 200 200 400 12800;
B 200 200 400 13500;
B 200 200 400 14200;
B 200 200 400 14900;
B 200 200 400 15600;
B 200 200 23000 10700;
B 200 200 21400 11400;
B 200 200 19800 13500;
B 200 200 18200 14200;
B 200 200 6500 12100;
B 200 200 4900 12800;
B 200 200 3300 14900;
B 200 200 1700 15600;

B 200 200 18600 -8000;
DF;

(Main of pma422.k);
C 15;
E

- 113 -

EXHIBIT

25

NCS Transmittal Form CB 00-13443 321

JSCC CSE Form No. 100 48 96 1 22

PLEASE DO NOT FOLD

UNIVERSITY OF SOUTH CAROLINA

FINAL REPORT OF GRADES
ALL STUDENTS

TOPICS/COMPUTER ENGR

SPRING 1986 SESSION-C002

SESSION DATES 012086-050586

SCHED. CODE 138586

04/20/86

EECE	890	001	KOBAYASHI	H
DEPT.	COURSE NO.	SECTION	INSTRUCTOR	

Student to provide information in this section. Do not enter name or grade in this section.

INSTRUCTOR'S SIGNATURE

STUDENT NO. NAME CREDIT GRADE

SCHOOL/MAJOR SPECIAL INFORMATION

1 435 452G AL-MOSSA A A 3G

2 435 452G ARAFEH AYMAN M 3G

3 435 452G CHENG GODFREY C 3G

4 435 452G CHUAH EU HOCK 3G

5 435 452G FOO YOON PIN 3G

6 435 452G JOO KI-HYUN 3G

7 435 452G LIM JIT YEW 3G

8 435 452G LIM MENG HIOT 3G

9 435 452G OZEKI TOORU 3G

10 435 452G PINTO MABIL B 3G

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

PLEASE MAKE NO EXTRANEIOUS MARKS BETWEEN THESE LINES

USE PENCIL TO FILL IN CIRCLE

MARK ONLY ONE GRADE

W P A B+ B C+ C D+ D F S U I T AUD

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

CONFIDENTIAL

SC 0003406

EXHIBIT

26

Project Description for ECE890B Spring 1986

Title: A Knowledge-based Behavioral Description Translator

Principal Investigator: Simon Foo

I. INTRODUCTION

Recent advances in "silicon compilation" have enabled designers to integrate algorithms and behavioral descriptions on custom VLSI chips. Full custom chips that used to require hundreds of man-hours in design time can now be completed in the order of several weeks. Several approaches have been suggested to translate high-level functional or behavioral descriptions to logic or transistor network description, or even geometry masks data [1-3].

This research explores a knowledge-based system (KBS) for mapping finite state descriptions to netlists for functional module interconnections. The front-end to the KBS is a rule-based specification description translator [4].

II. OBJECTIVE AND SIGNIFICANCE OF RESEARCH

The objective of this research is to implement a knowledge-base architecture translator for finite state descriptions. Several approaches will be considered in the course of this research.

This research will show how a combination of rules, backtracking, frames can be applied to translate behavioral descriptions directly to VLSI architectures.

III. BACKGROUND

SDF [5] is an intermediate form for describing finite state transitions, much similar to antecedent-consequence production rules. The objective is to completely specify the state diagrams for finite state machines (FSMs).

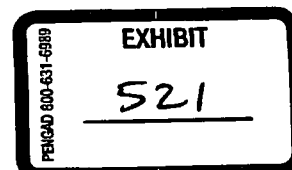
The syntax for SDF is as follows:

```
<rule> ::= <function> ':' <present state> {<condition list>}  
                                     <next state> {<action list>}
```

If <condition list> is TRUE, a transition is made from <present state> to <next state>. Otherwise, the function stays in <present state>.

SDF can be automatically generated by a specification description translator [].

CONFIDENTIAL



FOO 000189

IV. PREVIOUS WORK

A framework for managing VLSI CAD data has been implemented [6]. This frame-based database management system serves as a module (cell) library. Under this framework, whenever there is insufficient data, information can be inherited from design classes to its instances. The data base which is a collection of frames can be manipulated via a user interface. An example frame is as follows:

```
fput(M100; is_a "flip flop"; type "edge triggered";
      class "D"; tech "nmos 2.5"; status "tested";
      speed 15; power 100; width 6700; height 8540;
      inPorts "din100"; outPorts "Dq100, Dqn100";
      clock "clk100"; powerSource "vdd100, gnd100")
```

where "fput" puts information into the instance frame "M100".

V. OUTLINE OF PROPOSED FSM TRANSLATOR

This expert translator will consist of five major blocks and a knowledge base. These blocks serve to:

1. Make a state reduction and a state assignment.
2. Match a possible implementation given the next state code.
3. Select a set of functional modules to fulfill the specifications.
4. Evaluate the implementation constraints such as speed, area, etc.
5. Generate the interconnection netlists between modules.

OUTLINE OF PROPOSED WORK

As of today, most of the work on the knowledge base has been completed. The rest of the work will be done in the following order:

1. Implement the state reduction and state assignment routine.
2. Implement the "matcher" and "selection" routines.
3. Construct the "evaluation" routine using a backtracking approach.
4. Finally implement the netlist generator.

All the routines will be written in C. However, during the course of the research, the use of PROLOG to implement a

rule-based "evaluator" will be examined.

REFERENCES

1. J.M. Siskind, J.R. Southard, and K.W. Crouch, "Generating Custom High-performance VLSI Designs from Succinct Algorithmic Descriptions," MIT Conf. on Advanced Research in VLSI, Cambridge, MA, 1982.
2. D.J. Salomon, S. Sadler, and M.I. Elmasry, "A VLSI Architecture and a Silicon Compiler for Designing Numerical Processors," VLSI Design, Feb. 1985.
3. P.A. Subrahmanyam, "Synthesizing VLSI Circuits from Behavioral Specifications: a Very High level Silicon Compiler," VLSI 1983, F. Anceau, Ed., North Holland, Amsterdam 1983.
4. Y. Takefuji, M. Merx, D. Koutsourelis, "SDT: A Specification Description Translator," (under review).
5. H. Kobayashi, Y. Takefuji, "SDF: A State Description Form," Internal Report, Center for Machine Intelligence, University of South Carolina, Columbia, SC, 1985.
6. Y. P. Foo, and H. Kobayashi, "A Framework for Managing VLSI CAD Data," 1st International Conference on Applications of Artificial Intelligence to Engineering Problems, Southampton, England, April 1986.

EXHIBIT

27

A Framework for Managing VLSI CAD Data

Yoon-Pin Foo and Hideaki Kobayashi

*Dept. of Electrical and Computer Engineering, University of South Carolina,
Columbia,
SC 29208, U S.A.*

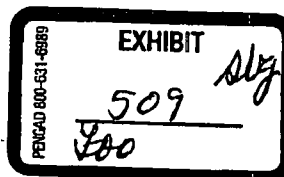
ABSTRACT

This paper presents a frame-based scheme for managing VLSI CAD data. The knowledge base for VLSI CAD is composed of a collection of design frames. The framework and inheritance algorithms are integrated into a frame manager (FAME). FAME has clear advantages over a relational schema in terms of speed, efficiency, and flexibility. The program is coded in C language and runs on a VAX-11/780 computer.

INTRODUCTION

There have been considerable interests in using commercial database management systems (DBMS) for managing VLSI CAD data. However, due to the complex nature of VLSI design data, most DBMS are not suitable for CAD applications [Katz83]. A dedicated DBMS is needed to support issues such as spatial searching, integrity constraints, design hierarchies, multilevel representations, and version control. Several approaches have been suggested as solutions to some of the above issues [StRG83], [Neum83], [Foo84].

This paper presents a frame-based scheme for managing VLSI CAD data. This approach is compared with an INGRES schema.



KBSC000904

890

FRAME-BASED SCHEME

The knowledge base for VLSI CAD consists of a collection of design objects. The key to managing the complexity of design data is to partition the knowledge base into small chunks of information each represented by a frame. A "design frame" is defined as a particular view of a design object. A tree structure of design frames completely describes a design object. Design frames can be categorized into "classes" and "instances". A "class" frame describes a generic function (e.g., multiplier), while an "instance" frame specifies a particular member of a class of design objects.

Figure 1 shows a set of design frames for an instance. The "is-a" (or "a-kind-of") slot contains the class membership information. The "owner-of" and "belong-to" slots specify other instances. The "module-id" slot contains a number which uniquely identifies the instance in a common knowledge base. The "version" slot contains the date of instance's creation or improvements. The "architecture", "logic", "circuit", and "layout" slots contain procedures to generate respective representations of the instance. Interconnections between instances are made via "terminals". The "type" slot indicates wire types (e.g., signal, power, and ground), while the "drive" slot indicates fan-out capabilities.

Since frames are association lists, they are implemented using a tree structure. Figure 2 shows a set of "design trees" which represents the design frames in Figure 1, where root nodes correspond to frame identifiers. Note that only VALUE facets are constructed as leaves.

Figure 3 shows a design tree for a class of instances. Note that only DEFAULT and other facets are constructed as leaves.

INFORMATION INHERITANCE

Based on the above framework, it is possible for a design instance to inherit information from its class. For example, properties of a design class (e.g., PLA) can be downward inherited to its instances through the "is-a" (or "AKO") slot. The VALUE inheritance procedure for a slot S in an instance node F is as follows:

KBSC000905

891

Form a queue Q consisting of all class nodes related to node F through the "is-a" slot.

```

IF Q is empty
  THEN announce failure
ELSE
  REPEAT
    IF the first element E of Q has a value
      in the VALUE facet of slot S
      THEN report the value and quit
      ELSE pop E from Q and add nodes
        found in the "is-a" slot of E to Q
  UNTIL a value is found or Q is empty.

```

Therefore, if there is no data in the VALUE facet of a particular slot, the VALUE, IF-NEEDED, and DEFAULT inheritance procedures are invoked successively to find a value.

If the above VALUE inheritance procedure fails, we can calculate a value using existing information. For example, without a value for the area of "pla382" (an instance of PLA), we can determine the area from its information on the number of inputs, products, and outputs. The function needed for such calculation is stuffed into the IF-NEEDED facet of the design class "PLA". The IF-NEEDED inheritance procedure is shown in Figure 4(a). The area-calculating function is inherited to "pla382" from the "PLA" class through the "is-a" slot. The inheritance procedure then creates an area slot for "pla382", and writes the calculated area into the VALUE facet, Figure 4(b).

The area-calculating function inherited to "pla382" is as follows:

```

IF there are values in the INPUTS, PRODUCTS, and
  OUTPUTS slots
  THEN ANDwidth = 16 * #INPUTS + 28
    ANDheight = 8 * #PRODUCTS + 49
    ANDarea = ANDwidth * ANDheight
    ORwidth = 8 * #OUTPUTS
    ORheight = 8 * #PRODUCTS + 68
    ORarea = ORwidth * ORheight
    AREA = ANDarea + ORarea
    Create an AREA slot for instance "pla382"
    and write the calculated area in the VALUE
    facet
  ELSE announce failure.

```

KBSC000906

892

PROGRAM IMPLEMENTATION

The frame-based scheme is implemented as shown in Figure 5. FAME consists of two major modules: a maintenance manager and a command interpreter.

To prevent unauthorized updates to FAME, the manager's functions are program embedded. Major tasks of the manager include manipulating design trees, handling of procedural attachments, and performing the inheritance procedures. More important, the manager validates the framework by checking for data inconsistencies (e.g., an instance being "owner-of" or "belong-to" itself).

The command interpreter provides a user interface to the manager. A comprehensive description of the interpreter, including the BNF, is given in [Foo85].

PERFORMANCE ANALYSIS

The performance of FAME is compared to a relational DBMS approach, with emphasis on speed, efficiency, and flexibility. The commercial relational DBMS benchmarked is INGRES. Consider the following relation from an INGRES schema for VLSI CAD data [Foo84]:

```
MODULE (module_id, module_name, designer, cif_id,
        status, function, technology)
```

Tuples can be appended to a "modules" relation using the following QUEL expression:

```
* append to modules (module_id = 236,
                      module_name = "pla382",
                      designer = "simon",
                      cif_id = 99,
                      status = "checked",
                      function = "full adder",
                      technology = "nmos/2.5")
```

Using FAME, the above "append" operation is performed in a set of subexpressions as follows:

```
APPEND(pla/pla382,module_id=236)
APPEND(pla/pla382,designer="simon")
APPEND(pla/pla382,cif_id=99)
APPEND(pla/pla382,status="checked")
APPEND(pla/pla382,function="full adder")
APPEND(pla/pla382,technology="nmos/2.5")
```

KBSC000907

Note that the order of each subexpression is not important since each subexpression is independent of others. More than one value can be inserted in a slot. For example, if the instance "pla382" is jointly designed, another subexpression is added:

```
APPEND(pla/pla382,designer="hideaki")
```

All of the above "incremental" subexpressions can be combined into a single APPEND expression:

```
APPEND(pla/pla382,module_id=236,designer="simon",
        designer="hideaki",cif_id=99,status="checked",
        function="full adder",technology="nmos/2.5")
```

FAME uses a hashing technique (where the key is the frame's identifier) to locate a desired frame. Then a breadth-first search is used to scan the slots. Since design trees are usually small, accessing a particular slot is fast and efficient.

An experiment is conducted to determine the relative speeds of FAME and an INGRES schema (without hashing and tree search techniques). The test data consists of 25 tuples from a collection of programmable logic cells described in [KoFo84]. Table I summarizes the average CPU time per tuple required for each operation.

TABLE I. BENCHMARK TESTS IN CPU MILLISECONDS

Operation type	FAME	INGRES
append	15	41
retrieve	16	42
update	18	40
delete	14	38

In summary, the advantages of FAME over INGRES are as follows:

1. FAME downloads the knowledge base into the virtual memory of a VAX-11/780 computer using its unique data structure. INGRES stores the design data as disk resident relations with only small portions in a main memory.
2. FAME uses a dynamic tree structure where new frames, slots, and facets can be easily created and manipulated. Relations in an INGRES schema have predefined structures.

894

CONCLUSION

A frame-based scheme for managing VLSI CAD data has been presented. Results show that FAME, using a dedicated data structure, outperforms a relational schema. Future extensions to FAME include an interface to an inference engine for VLSI CAD applications.

REFERENCES

Foo, Y. P. (1984) Managing VLSI Design Data with a Relational Database System. M.S. Thesis. Dept. of Electrical and Computer Engineering, University of South Carolina, Columbia, SC.

Foo, Y. P. (1985) FAME: A User's Manual. Internal Technical Report. Dept. of Electrical and Computer Engineering, University of South Carolina, Columbia, SC.

Katz, R. H. (1983) Managing the Chip Design Database. Computer, vol. 16, no. 12, pp. 26-35.

Kobayashi, H. and Foo, Y. P. (1984) Programmable Logic for Parallel Convolution. Int. Conf. Computer Design: VLSI in Computers (ICCD), Port Chester, NY, Oct., pp. 700-704.

Neuman, T. (1983) On Representing the Design Information in a Common Database. Engineering Design Applications Proc. Annual Meeting - Database Week, San Jose, CA, May, pp. 81-87.

Stonebraker, M., Rubenstein, B., Guttmann, A. (1983) Application of Abstract Data Types and Abstract Indices to CAD Data Bases. Engineering Design Applications Proc. Annual Meeting - Database Week, San Jose, CA, May, pp. 107-113.

KBSC000909

895

Design Instance	documentation
is-a	representations
owner-of	constraints
belongs-to	terminals

Instance Doc	designer
module-id	version
function	status

Instance Constraints	
technology	area
speed	power

Instance Rep	
architecture	circuit
logic	layout

Instance Terminal	owner
drive	type

Figure 1. A set of design frames for an instance.

KBSC000910

896

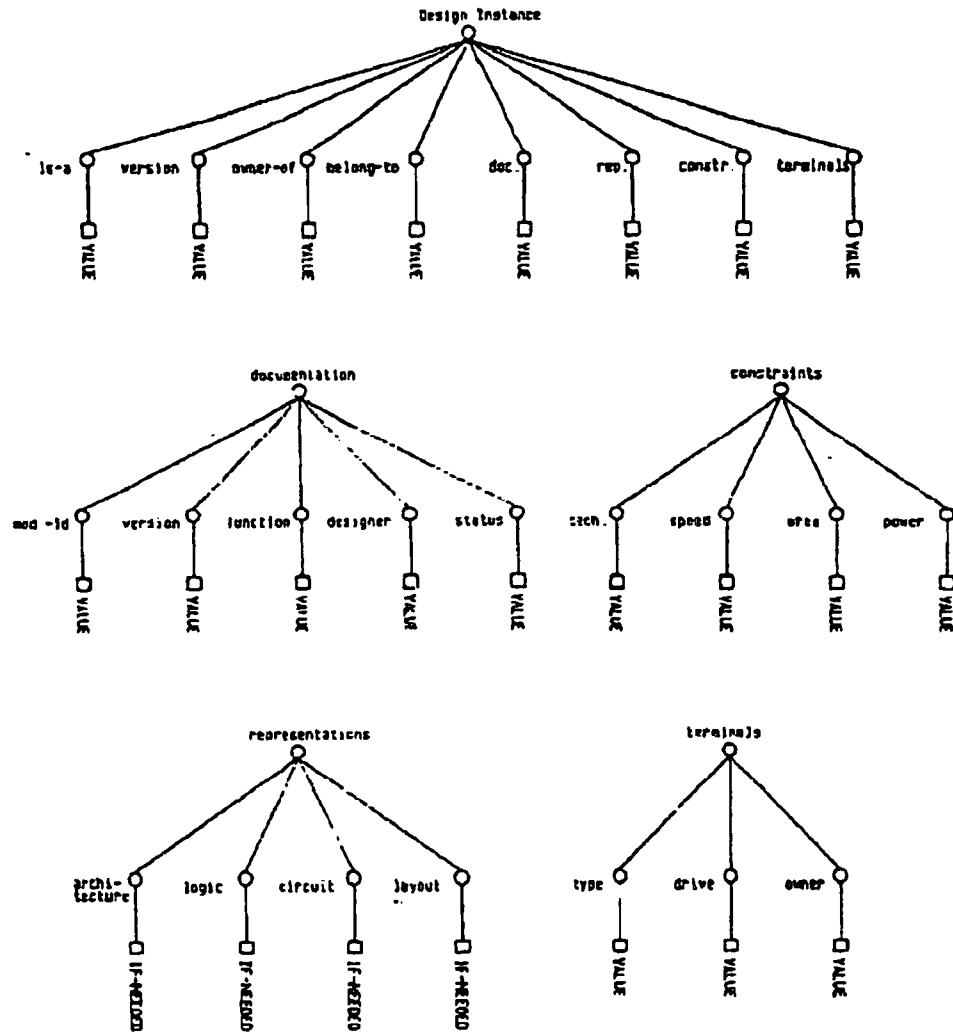


Figure 2. A set of design trees for an instance.

KBSC000911

897

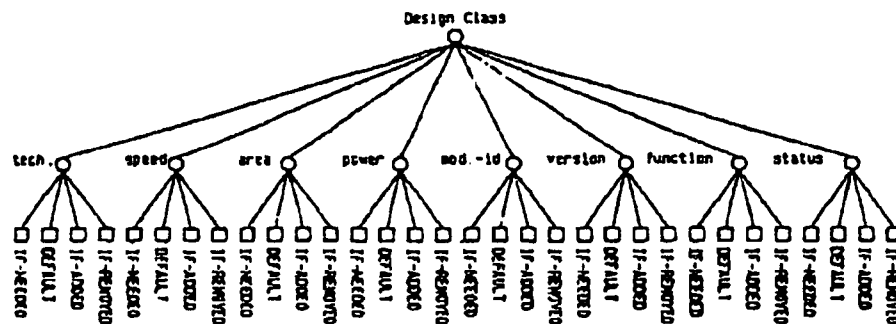
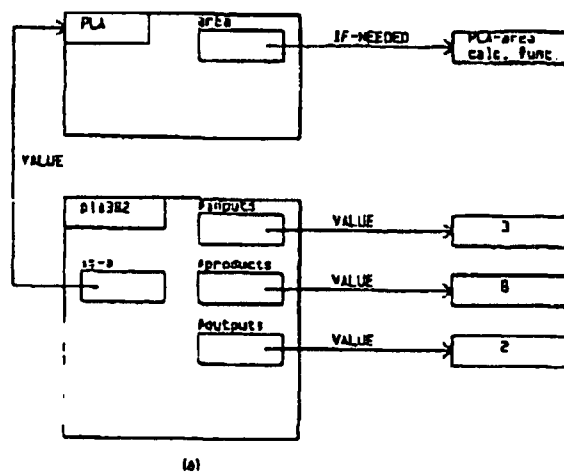
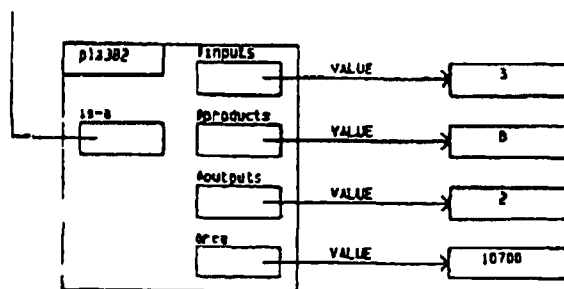


Figure 3. A design tree for a class of instances.



(a)



(b)

Figure 4. An IF-NEEDED inheritance procedure.

KBSC000912

895

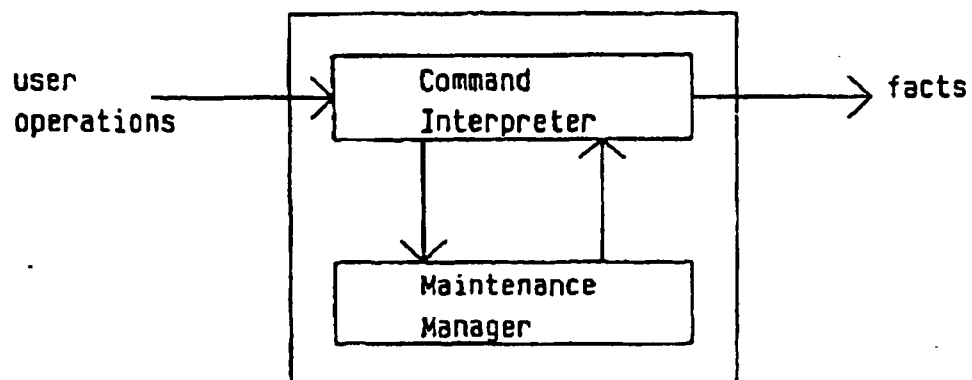


Figure 5. Configuration of FAME.

KBSC000913

EXHIBIT

28

FEB 26 2002 11:33 PM FR NRC

00000 TO 12027752593

P.03

A Knowledge-Based System for VLSI Module Selection

Yoon-Pin Simon Foo and Hideaki Kobayashi

Center for Machine Intelligence
Department of Electrical and Computer Engineering
University of South Carolina
Columbia, SC 29208
(803)777-6953

ABSTRACT

Heuristic rules for handling implementation alternatives are integrated into a frame-based system for selecting functional modules. An inheritance mechanism is embedded in a framework for managing VLSI CAD data. An optimized module set is obtained using a rank-and-sort routine and a dependency-directed backtracking algorithm.

INTRODUCTION

Recent research in exploratory VLSI design environments includes PALLADIO[1] and CMUDA[2]. However, these tools lack a module selection mechanism. Currently, the time consuming process of module selection is made by expert designers with the help of circuit simulators and timing verifiers. An expert or knowledge-based system is needed to perform efficient decision-making in a large VLSI module library.

This paper introduces a frame-based system for selecting VLSI modules, called NEPTUNE. Based on domain specific knowledge and heuristic rules, NEPTUNE assists IC designers to select an optimized solution, and explore different implementation alternatives.

Expert knowledge for VLSI CAD including implementation alternatives and design trade-offs are presented. A rank-and-sort routine and a backtracking algorithm for evaluating modules are presented. An example selection of a module set to implement a digital filter is demonstrated.

KNOWLEDGE REPRESENTATION

Information for module selection is provided by a domain specific database implemented on a framework for representing VLSI CAD data [3]. The database consists of *design objects* which can be classified into two types: *simple modules* with single control state (NAND, NOR, XOR, multiplexer, etc.), and *complex modules* with multiple control states (counter, system controller, data-path, etc.). Design objects can be *classes* or *instances*. Each design object is represented by a *frame*.

This work is supported by International Chip Corporation

Based on the above framework, VLSI CAD data is represented as a hierarchy of *design frames*. For example, the module hierarchy of an 8-to-1 multiplexer is represented as follows:

```
fpur(mux2; is_a "multiplexer"; bit_size 2;
           components "nand,nand,inverter")
fpur(mux4; is_a "multiplexer"; bit_size 4;
           components "mux2,mux2,mux2")
fpur(mux16; is_a "multiplexer"; bit_size 16;
           components "mux4,mux4,mux4,mux4")
```

where *fpur* appends information to database.

If needed, information can be inherited from design classes to instances through *semantic links (slots)* such as *is_a*, or a *kind of (AKO)*. This inheritance mechanism is a set of rules to make deductive inferences on a domain specific database. A subset of the mechanism is as follows:

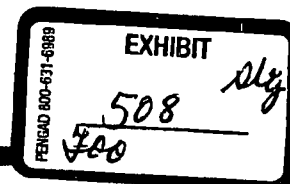
```
IF (!function) THEN inherit function through semantic links;
IF function is comparator THEN number of outputs is 3;
IF status is tested THEN status of submodules is tested;
IF technology is CMOS THEN
    technology of submodules is CMOS.
```

where attribute *status* specifies whether a module is functionally tested, logically simulated, or in a development stage.

IMPLEMENTATION ALTERNATIVES

Implementation alternatives are important since they allow designers to explore new chip designs using different module structures with varying characteristics. For example, a logic function can be implemented in various structures, with the choice of several driver buffers and positions of I/O ports. Figure 1 shows a set of possible CMOS structures for implementing XOR function (i.e., $F = A \oplus B = A \oplus B$). Each configuration exhibits a different set of implementation constraints such as speed, area, and power consumption. These statistics extracted from circuit simulators (e.g., SPICE[4]) and timing verifiers (e.g., CRYSTAL[5]) are appended to the design database.

Heuristic rules are needed to resolve design trade-offs, if design constraints are not satisfied. For example, a carry-ripple adder may be advantageous over a carry-look-ahead adder if smaller area is desired. Similarly, vice-versa is true if speed is critical.



FEB 26 2002 11:33 PM FR NRC

00000 TO 12027752593

P.04

In the absence of implementation alternatives and design trade-offs, a list of logic components (submodules) necessary to implement the function is provided. For example, the design database has structural information about XOR (as shown in Figure 1):

```
spur(xor1: is_a "xor"; components "inverter,inverter,
                                ntransistor,ntransistor")
spur(xor2: is_a "xor"; components "nand,nand,nand,nand")
spur(xor3: is_a "xor"; components "PLA")
```

By default, a module with simplest logic structure is chosen.

SELECTION MECHANISM

The selection of modules depends on the goals (or design constraints) associated with particular implementation alternatives (random logic, PLA, etc.) If there is no instance in the module library that satisfies particular constraint, then NEPTUNE selects a design alternative (if any), or outputs a list of logic components if everything else fails.

Figure 2 shows the architecture of NEPTUNE where the input file is a list of design specifications (bit size, technology, etc.) associated with each module. Selector uses a heuristic search technique to return a list of instances whose *is_a* (or *ARO*) slots are *subfunction* and satisfy the design constraints.

Evaluator then invokes a dependency-directed backtracking routine to weigh instances according to design constraints (speed, area, and power, etc.). The evaluation process is supported by user entered information on path delay, area, and power analysis. These statistics are quantized to a scale from 1 to 10 (least to most critical). If needed, a procedure to calculate the bounding area of module is invoked.

The module selection mechanism, incorporating a heuristic search technique and a backtracking algorithm, can be abstracted as follows:

- 1) Initialize selected module list S.
- 2) For each subfunction, modules which satisfy design specifications are grouped into subfunction choice lists.
- 3) For each *empty* subfunction choice list, IF subfunction has a list of logic components THEN append list to S, and notify user; ELSE announce failure for subfunction.
- 4) For each *non-empty* choice list, rank and sort instances according to speed, area, and power.
- 5) Create working module list W, consisting of the first instances from each *non-empty* subfunction choice list.
- 6) REPEAT for each instance in W, IF (speed & area & power are satisfied) THEN append instance to S; /* Rules to handle design trade-offs */ ELSE IF (speed & area are satisfied) THEN append instance to S, and notify user; ELSE IF (speed & power are satisfied) THEN append instance to S, and notify user; ELSE IF (speed is satisfied) THEN append instance to S, and notify user; ELSE subfunction is tagged *problem*.

7) For each *problem* subfunction, IF choice list is *empty* THEN GOTO step 9; ELSE choose a new instance from choice list, and update W.

8) REPEAT step 6.

9) REPEAT for each *problem* subfunction, /* Resolve the default list of submodules */ IF subfunction has lists of submodules THEN append to S the list with most simple modules; ELSE announce failure for particular subfunction.

In the event of a tie, module which is most specialized (i.e., with the longest list of information slots) is selected.

DESIGN EXAMPLE

Consider the basic structure of a digital nonrecursive filter (DNF). Figure 3 Minimal design specifications to implement a DNF are:

```
begin_select
fget(register; where bit_size=8 & technology="cmos")
fget(register; where bit_size=8 & technology="cmos")
fget(register; where bit_size=20 & technology="cmos")
fget(comparator; where bit_size=4 & technology="cmos")
fget(multiplier; where bit_size=8 & technology="cmos")
fget(counter; where bit_size=4 & type="synchronous"
               & technology="cmos")
spur(index; delay 5; area 5; power 5)
end_select
```

where *index* frame specifies the desired rankings of design constraints. If performance indices are not specified, NEPTUNE outputs the first solution set with optimum speed, area, and power consumption. Notice the system controller is generated independently.

PERFORMANCE ANALYSIS

If the user's input specifications are more *specialized* (i.e., with a longer list of constraints), then the choice set for each subfunction will have less number of candidates. Therefore the speed of the selection mechanism is directly proportional to the number of constraints. Hence, it is desirable to have as much specializations as possible to restrict the search space.

Since the modules are pre-sorted before evaluation, minimum backtracking is achieved. A quick-sort routine is utilized for maximum speed.

NEPTUNE is coded in C, and runs on a VAX-11/780 computer and SUN workstations. Figure 5 shows average CPU access time on a VAX-11/780 for selecting a set of modules to implement a DNF in a design database. Notice the access time increases linearly with the number of modules. The linearity is due to a combination of hashing, quick-sorting, and minimal backtracking.

FEB 26 2002 11:34 PM FR NRC

00000 TO 12027752593

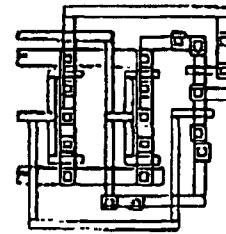
P.05

CONCLUSION

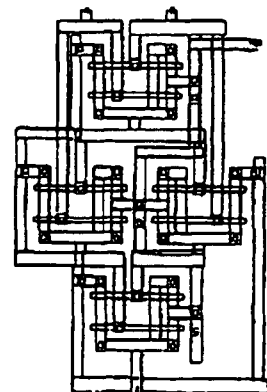
In an overall framework for selecting VLSI modules, this paper has provided an inheritance mechanism, rules for handling implementation alternatives and design trade-offs, with minimal backtracking. As a design assistant tool, *NEPTUNE* is indispensable during a VLSI feasibility study since it eliminates exhaustive search in a design database. Performance analysis showed that *NEPTUNE* is fast and efficient for large-scale databases. Future extensions include porting *NEPTUNE* to a distributed environment, and interfacing to existing VLSI CAD tools.

REFERENCES

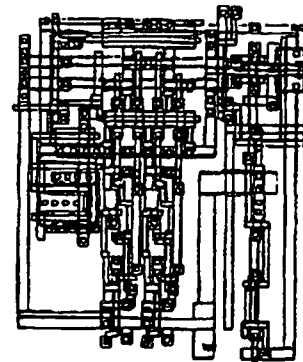
- [1] H. Brown, C. Tong, and G. Foyster, "Palladio: An Exploratory Environment for Circuit Design," Computer, Dec. 1983.
- [2] T.J. Kowalski, and D.E. Thomas, "The VLSI Design Automation Assistant: Prototype System," Proc. 20th Design Automation Conf., June 1983.
- [3] Y.P. Foo, and H. Kobayashi, "A Framework for Managing VLSI CAD Data," 1st Int. Conf. on Applications of Artificial Intelligence to Engineering Problems, Southampton, England, April 1986.
- [4] A. Vladimirescu, A.R. Newton, and D.O. Pederson, "SPICE Version 2G.1 User's Guide," Dept. of Electrical and Computer Sciences, Univ. of California, Berkeley, CA, Oct. 1980.
- [5] J. Ousterhout, "Using Crystal for Timing Analysis," Report No. UCB/CSD 86/272, University of California, Berkeley, CA, Dec. 1985.



(a) Pass transistor logic.



(b) NAND-NAND logic.



(c) PLA.

Figure 1. Implementation alternatives for XOR.

FEB 26 2002 11:34 PM FR NRC

00000 TO 12027752593

P.06

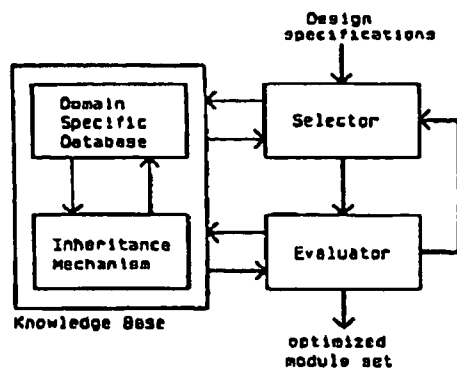


Figure 2. Architecture for a knowledge-based module selector

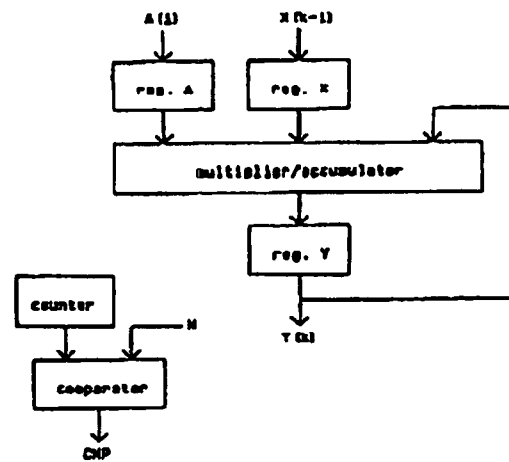


Figure 3. Functional modules to implement a digital filter.

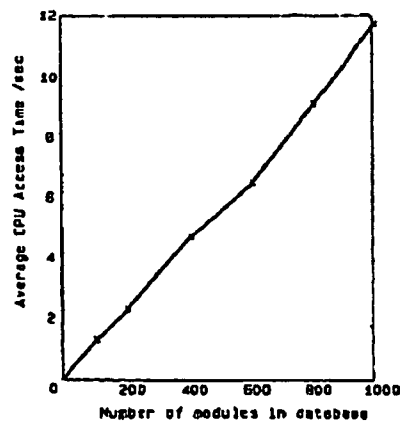


Figure 4. Average CPU access time for NEPTUNE.

EXHIBIT

34

Exhibit A ('432 Patent)

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising	<p>During manufacture of a desired application specific integrated circuit (ASIC) chip that is designed to perform a specific purpose, a process of designing the desired ASIC using a computer, the process comprising:</p> <p>("application specific integrated circuit (ASIC)"= an integrated circuit chip designed to perform a specific function.)</p> <p><u>Support</u></p> <p>'432 Patent: Column 1, lines 13-17; column 2, lines 15-20.</p> <p>RCL000207-223.</p> <p>"computer-aided design": The use of computers to aid in design layout and analysis. IEEE Standard Dictionary of Electrical and Electronic Terms,</p>	<p>A. "A computer-aided design process for designing" -- a process that uses a computer for designing, as distinguished from a computer-aided manufacturing process, which uses a computer to direct and control the manufacturing process.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: 1:9-12; 16:34-65 [Attachment 2¹, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶¶ 16-18²;</p> <p>CAD Tool Integration For ASIC Design: at 364-365 [Attachment 21, KBSC000031-KBSC000036].</p> <p><i>Dictionaries, Treatises, Textbooks, etc.:</i></p> <p>IBM Dictionary of Computing: at 129-130 [Attachment 16, DEF083932-DEF084703]; IC Mask Design: at 1-24 Attachment 17,</p>

¹ Referenced attachments are attached to this Exhibit A.

² "Decl. Kowalski ¶¶" refers to the pertinent paragraphs in the Declaration and Summary of Opinions of Dr. Thaddeus J. Kowalski on Construction of Disputed Claim Terms of United States Patent No. 4,922,432 that support Synopsys' and Defendants' proposed constructions and oppose Ricoh's proposed constructions for the disputed claim term, phrase or clause. Ricoh objects to the citation of extrinsic evidence by Synopsys and the Aeroflex defendants, and reserves its right to seek discovery regarding any such extrinsic evidence and respond with its own extrinsic evidence.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
	<p>Fourth Edition (1988). RCL011382-388 at RCL011384.</p> <p>“function”: A specific purpose of an entity or its characteristic action. IEEE Standard Dictionary of Electrical and Electronic Terms, Fourth Edition (1988). RCL011382-388 at RCL011386.</p> <p>“function”: Any of a group of related actions contributing to a larger action. Merriam-Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011400.</p>	<p>DEF085303-DEF085329]; Microchip Fabrication: at 76-82, 274-278 [Attachment 18, DEF085330-085343]; IEEE Standard Dictionary of Electrical and Electronics Terms: at 180 [Attachment 19, DEF085299-085302]; Webster's Dictionary: at 343, 725 [Attachment 20, DEF085290-DEF085298].</p> <p>B. “application specific integrated circuit” -- an interconnected miniaturized electronic circuit on a single piece of semiconductor material designed to perform a specific function, as distinguished from standard, general purpose integrated circuits, such as microprocessors, memory chips, etc.</p> <p><i>Intrinsic Evidence</i>: ‘432 patent: 1:13-17; 16:34-65 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence</i>: Decl. Kowalski ¶¶ 19-20; CAD Tool Integration For ASIC Design: at 363 [Attachment 21, KBSC000031-KBSC000036].</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
<p>storing a set of definitions of architecture independent actions and conditions;</p>	<p>Placing in computer memory a library of definitions of the different architecture independent actions and conditions that can be selected for use in the desired ASIC, where the architecture independent actions and conditions do not imply any structure or implementing technology.</p> <p>("architecture independent action and condition"= functional or behavioral aspects of a portion of a circuit (or circuit segment) that does not imply any set architecture, structure or implementing technology.)</p> <p><u>Support</u></p> <p>'432 Patent: Figs. 1a, 1b, 1c, 4; column 2, lines 6-20; column 2, lines 27-34; column 3, line 49 to column 4, line 4; column 4, lines 5-19; column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 10, lines 10-12; column 13, lines 2-31.</p>	<p>C. ³ "actions and conditions"-- are the logical steps and decisions that are represented as rectangles and diamonds in the flowchart; collectively logical operations.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: 2:24-27; 3:50-59; 4:15-19; 4:61-63; 6:3-14; 7:20-23; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶ 21.</p> <p><i>Dictionaries, Treatises, Textbooks, etc.:</i></p> <p>IBM Dictionary of Computing: at 479 [Attachment 16, DEF083932-DEF084703].</p>

³ Synopsys and the Defendants disagree with Ricoh's definition of "storing" on this step and the next two steps as "placing in computer memory." Storing means placing on any storage device "that is accessible by the processor for the computer system." IBM Dictionary of Computing at 654 (Attachment 16, DEF083932-DEF084703).

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
	<p>“action”: A thing done. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011391.</p> <p>“architecture”: A unifying or coherent form or structure. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011393.</p> <p>“independent”: Not dependent; not requiring or relying on something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011394.</p> <p>“condition”: Something essential to the appearance or occurrence of something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405A</p>	<p>D. “architecture independent”⁴ -- not including (i.e., excluding) a register transfer level (RTL) description or any other description that is hardware architecture dependent. An RTL description consists of: 1) defining the inputs, outputs, and any registers of the proposed ASIC; and, 2) describing for a single clock cycle of the ASIC how the ASIC outputs and any registers are set according to the values of the ASIC inputs and the previous values of the registers; an RTL description defines any control needed for the ASIC.</p> <p><i>Intrinsic Evidence:</i> ‘432 patent file history: April 1989 Amendment at 8-10, 13; November 1989 Amendment at 7 [Attachment 3, DEF011820-DEF012110]; ‘435 patent: Fig. 4; 4:26-32; 5:27-35 [Attachment 4, DEF012503-DEF012518].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 22-26; Computer Aided VLSI Design Vol.1 No. 4: at 388</p>

⁴ This phrase was not in the original patent application and therefore violates the prohibition of Section 132 of 35 U.S.C. against adding new matter. Moreover, it is indefinite and inadequately described in the ‘432 patent. The only description is in the ‘432 patent's file history, which is in the negative.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>[Attachment 22, KBSC001109-KBSC001131].</p> <p>E. "a set of definitions of architecture independent actions and conditions" -- a set of named descriptions defining the functionality and arguments for the available logical steps and decisions that may be specified in the flowchart; and excluding a register transfer level description.</p> <p><i>Intrinsic Evidence:</i> '432 patent: 4:61-63; 5:20-22; 6:3-14; 7:25-50; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 27-28.</p>
storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;	<p>Placing in computer memory a library of cell information that describe hardware cells capable of performing the different architecture independent actions and conditions placed in the library of definitions.</p> <p>("hardware cells"= previously designed circuit components or structure that have specific physical and functional characteristics used as building blocks for implementing an ASIC to be</p>	<p>F. "hardware cells" -- logic blocks for which the functional level (e.g., register transfer level), logic level (e.g., flip flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) descriptions are all defined.</p> <p><i>Intrinsic Evidence:</i> '432 patent: 2:34-39; 3:59-67; 5:15-20; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis's & Defendants' Construction(s) And Support For Those Construction(s)
	<p>manufactured.)</p> <p><u>Support</u></p> <p>'432 Patent: Fig. 4; column 2, lines 36-39; column 4, line 68 to column 5, line 3; column 5, lines 15-20; column 5, lines 22-25; column 9, lines 24-60; column 10, lines 10-12.</p>	<p>DEF012585]; '016 patent: 11:47-13:63; 14:3-22 [Attachment 5, DEF017265-DEF017284].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶¶ 29-30;</p> <p>Computer Aided VLSI Design Vol.1 No. 4: at 380 [Attachment 22, KBSC001109-KBSC001131].</p> <p>G. "data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set" -- a set of named integrated circuit hardware cells that includes at least one hardware cell for each stored definition that may be specified for the available logical steps and decisions; where each named hardware cell has corresponding descriptions at the functional level (e.g., register transfer level), logic level (e.g., flip-flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) that are all defined.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: Fig. 4; 2:34-39; 3:59-67; 5:15-20; 5:23-25; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i></p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;	Placing in an expert system knowledge base, that uses a computer memory, a plurality of rules for selecting among the hardware cells placed in the hardware cell library, wherein the rules comprise the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures. ("expert system knowledge base"= database used to store expert knowledge of highly skilled VLSI designers.) ("rules"= the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures.) <u>Support</u> '432 Patent: Column 5, lines 6-8; column 8, line 65 to column 9, line 5; column 9, lines 14-20; column 10, lines 1-10; column 10, line 40 to column 11, line 14; column 11, lines 16-26; column 11, lines 30-32; column 11, line 46 to column 12, line 35. RCL000229-237.	Decl. Kowalski ¶¶ 31-32. H. "expert system" -- software executing on a computer system that attempts to embody the knowledge of a human expert in a particular field and then uses that knowledge to simulate the reasoning of such an expert to solve problems in that field. This system is comprised of a knowledge base containing rules, working memory containing the problem description, and an inference engine. It solves problems through the selective application of the rules in the knowledge base to the problem description, as distinguished from conventional software, which uses a predefined step-by-step procedure (algorithm) to solve problems. <i>Intrinsic evidence:</i> '432 patent: 2:58-63; 5:6-8; 8:58-60; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '432 patent's file history: April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-012110]; '016 patent: 2:65-3:8 [Attachment 5, DEF017265-DEF017284]; '435 patent: 7:32-9:35[Attachment 4, DEF012503-

Claim Element	Rico's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
	<p>"rule": A prescribed guide for conduct or action; an accepted procedure, custom or habit. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405.</p> <p>"expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410.</p>	<p>DEF012518]; '603 patent: 1:44-49 [Attachment 6, DEF017456-DEF017482]; An Overview of Logic Synthesis Systems: at 170 [Attachment 7, DEF011962-DEF011968]; The CMU Design Automation System: at 75-77 [Attachment 8, DEF012045-DEF012052].</p> <p>I. "Knowledge base" -- the portion of the expert system containing a set of rules embodying the expert knowledge for the particular field.</p> <p><i>Intrinsic evidence:</i></p> <p>'432 patent: 10:39-11:15; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012864-DEF012585]; '432 patent file history: April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; '435 patent: 7:32-9:35 [Attachment 4, DEF012503-DEF012518].</p> <p>J. "a set of rules for selecting hardware cells to perform the actions and conditions" -- a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis's & Defendants' Construction(s) And Support For Those Construction(s)
		<p>expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description.</p> <p><i>Intrinsic evidence:</i></p> <p>'432 patent: 2:58-63; 8:20-30; 8:58-9:62; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; '435 patent: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; An Overview of Logic Synthesis Systems: at 170 [Attachment 7, DEF011962-DEF011968]; The CMU Design Automation System: at 75-77 [Attachment 8, DEF012045-DEF012052].</p> <p><i>Extrinsic Evidence for all of the above phrases (H, I, & J) in this step:</i></p> <p>Decl. Kowalski ¶¶ 33-45; Computer Aided VLSI Design Vol.1 No. 4: 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131]; '669 patent: Abstract, 1:13-16, 2:28-33, 4:31-62; 5:21-38, 5:58-68, 6:1-7:68, 17:62 [Attachment 23, DEF 080579-DEF080605]; The VLSI</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis's & Defendants' Construction(s) And Support For Those Construction(s)
		<p>Design Automation Assistant: at 889-90 [Attachment 28, DEF018660-DEF018664]; A Rule-Based Logic Circuit Synthesis System for CMOS Gate Arrays: at 597 [Attachment 29, DEF018277-DEF018283].</p> <p><i>Dictionaries, Treatises, Textbooks, etc. for all of the above phrases (H, I, & J in this step):</i></p> <p>Understanding Expert Systems: 7-10, 29-30, 40, 42, 74-78, 99-110 [Attachment 24, DEF079512-DEF079741]; An Artificial Intelligence Approach To VLSI Design: at 9-15 [Attachment 25, DEF078425-DEF078455]; Artificial Intelligence Terminology: at 6 [algorithm], 10 [antecedent], 53 [consequent], 86-87 [expert system], 127 [inference engine], 140 [knowledge based system], 204 [production system], 223 [rule base, rule-based system], 281 [working memory] [Attachment 26, DEF079212-DEF079511]; Microsoft Press Computer Dictionary: at 136 [expert system] [Attachment 27, DEF083323-DEF083325]; IBM Dictionary of Computing: 591 [rule interpreter, rule-based system] [Attachment 16, DEF083932-DEF084703]; Expert Systems: A Non-Programmer's Guide: 8-10, 16 [Attachment 30, DEF082264-DEF082528]; Expert Systems:</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis's & Defendants' Construction(s) And Support For Those Construction(s)
		<p>Tools and Applications: at 269 [Attachment 31, DEF079985-DEF080283]; Expert Systems: Principles and Case Studies: at 11-12 [Attachment 32, DEF079753-DEF079984]; Knowledge-Based Systems: The View in 1986: at 16 [Attachment 33, DEF083251-DEF083284].</p>
<p>describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;</p>	<p>A user describing an input specification containing the desired functions to be performed by the desired ASIC.</p> <p><u>Support</u></p> <p>'432 Patent: Column 1, line 13 to column 2, line 3, column 2, lines 6-20, column 2, lines 21-24; column 2, lines 27-34; column 6, lines 58-60; column 13, lines 32-35; column 14, lines 7-29.</p> <p>RCL000207-223, RCL000229-237.</p> <p>"describe": To represent or give an account in words <~ a picture>; to represent by a figure, model, or picture: delineate. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398.</p> <p>"description": A descriptive statement or account.</p>	<p>K. "describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions" -- the designer represents a sequence of logical steps (rectangles) and decisions (diamonds), and the transitions (lines with arrows) between them in a flowchart format for a proposed application specific integrated circuit.</p> <p><i>Intrinsic evidence:</i></p> <p>'432 patent: Figs. 1a, 5, & 7; 2:21-27; 3:20-22; 3:50-59; 4:5-22; 4:35-38; 7:12-23; 16:34-65 [Attachment 2, DEF012564-012585]; '432 patent file history: April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110]; '016 patent: 7:33-9:52 [Attachment 5, DEF017265-DEF017284]; '435 patent: 4:26-33 [Attachment 4, DEF012503-DEF012518]; Flamel: A High-</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
	<p>Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398.</p>	<p>Level Hardware Compiler: 260-261 [Attachment 9, DEF012034-DEF012044]; An Overview of Logic Synthesis Systems: at 168-170 [Attachment 7, DEF011962-DEF011968]; Methods Used in an Automatic Logic Design Generator (ALERT): at 595 [Attachment 10, DEF011969-DEF011989]; Quality of Designs From an Automatic Logic Generator (ALERT): at 71 [Attachment 11, DEF012005-DEF012023]; The CMU Design Automation System: at 73-74 [Attachment 8, DEF012045-DEF012052]; A New Look at Logic Synthesis: at 544 [Attachment 12, DEF012024-DEF012030]; Experiments in Logic Synthesis: at 235 [Attachment 13, DEF011990-DEF011994]; CAD Systems for IC Design: at 3, 7 [Attachment 14, DEF011951-DEF011961]; Verifying Compiled Silicon: at 2 [Attachment 15, DEF011948-DEF011950]; '442 patent: 5:3-46, [Attachment 35, DEF012392-DEF012401; '603 patent: 2:41-61 [Attachment 6, DEF012392-DEF012401].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 46-49; Computer Aided VLSI Design Vol.1 No. 4: 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131];</p>

Claim Element	Ricoli's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>Silicon Compilation: at 48-49 [Attachment 34, DEF076335-DEF076341].</p> <p><i>Dictionaries, Treatises, Textbooks, etc.:</i> Webster's Ninth New Collegiate Dictionary at 343, 1074, 1073 [Attachment 20, DEF085290-DEF085298].</p>
specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and	<p>Specifying for each desired function to be performed by the desired ASIC one of the definitions of the architecture independent actions and conditions stored in the library of definitions that is associated with the desired function.</p> <p>("specifying"= mapping or associating a desired function to be performed by the manufactured ASIC with a definition from the library of definitions.)</p> <p><u>Support</u></p> <p>'432 Patent: Column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 8, lines 65-67; column 9, lines 8-18; column 10, lines 10-12; column 13, lines 2-31.</p>	<p>L. "specifying for each described action and condition of the series one of said stored definitions" -- the designer assigns one definition from the set of stored definitions for each of the described logical steps and decisions represented in the flowchart.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '016 patent: 6:12-32 [Attachment 5, DEF017265-DEF017284].</p> <p><i>Dictionaries, Treatises, Textbooks, etc.</i> Webster's Ninth New Collegiate Dictionary at 1132 [Attachment 20, DEF085290-DEF085298].</p> <p>M. "which corresponds to the desired action or condition to be performed" -- each specified definition must correspond to the intended step or</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>decision to be performed.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '016 patent: 6:12-32 [Attachment 5, DEF017265-DEF017284].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 50-52.</p> <p><i>Dictionaries, Treatises, Textbooks, etc.:</i> Webster's Ninth New Collegiate Dictionary at 110 [Attachment 20, DEF085290-DEF085298].</p>
<p>selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware</p>	<p>Selecting from the plurality of hardware cells in the hardware cell library a hardware cell for performing the desired function of the desired ASIC through application of the rules; and generating a netlist that identifies the hardware cells needed to perform the function of the desired ASIC and the necessary parameters for connecting the respective inputs and outputs of each hardware cell, the netlist is passed to the next subsequent step in the process for manufacturing the desired ASIC.</p> <p>("netlist"= a description of the hardware components</p>	<p>Synopsys and Defendants provide their constructions for the disputed claim terms, phrases, and clauses and support for these two separate "selecting" and "generating for the selected integrated hardware cells" steps below and separately for each step. Synopsys and Defendants dispute Ricoh's attempt to improperly and misleadingly combine these two separate steps.</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
<p>cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.</p>	<p>(and their interconnections) needed to manufacture the ASIC as used by subsequent processes, e.g., mask development, foundry, etc.)</p> <p><u>Support</u></p> <p>'432 Patent: Figs. 4, 9, 13; column 1, lines 17-26; column 2, lines 34-36; column 2, lines 42-44; column 5, lines 25-29; column 5, lines 35-40; column 8, lines 21-23; column 8, lines 26-41; column 8, lines 58-64; column 9, lines 8-24; column 9, line 64 to column 10, line 7; column 10, lines 13-34; column 13, line 36 to column 14, line 6.</p> <p>RCL000207-223, RCL000229-237.</p> <p>"expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410.</p> <p>"interconnection": To connect with one another. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011403.</p>	

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
	<p>"requirement": Something wanted or needed- Necessity. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011404.</p>	
<p>selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert</p>	<p>See the "selecting" row at column 2, <i>supra</i>, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.</p>	<p>N. "selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit" -- mapping the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Fig. 4; 3:16-19; 4:66-5:3; 5:22-29; 8:31-37; 8:58-60; 9:52-60; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 10 [Attachment 3, DEF011820-DEF012110].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶ 55.</p> <p>O. "said step of selecting a hardware cell comprising applying to the specified definition of the action</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
system knowledge base and		<p>or condition to be performed, a set of cell selection rules stored in said expert system knowledge base" -- the mapping of the specified definitions to the stored hardware cell descriptions must be performed by an expert system having an inference engine for selectively applying a set of rules, each rule having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Abstract; 2:58-63; 5:6-8; 8:29-37; 8:58-60; 9:8-13; 11:16-26; 16:34-65 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 8-11, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 4, 6-7, 9 [¶¶] Attachment 3, DEF011820-DEF012110]; '435 patent: 7:32-9:35[Attachment 4, DEF012503-DEF012518]; '603 patent: at 3:59-63 [Attachment 6, DEF017456-DF017482]; '016 patent: 3:5-8; 9:67-10:2, [Attachment 5,</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>DEF017265-DEF017284; An Overview of Logic Synthesis Systems: at 170 [Attachment 7, DEF011962-DEF011968]; The CMU Design Automation System: at 75-77 [Attachment 8, DEF012045-DEF012052].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶¶ 56-57</p> <p>Same as set forth for phrases H, I, & J above.</p> <p><i>Dictionaries, Treatises, Textbooks, etc.</i></p> <p>Same as set forth for phrases H, I & J above.</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
<p>generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.</p>	<p>See the "selecting" row at column 2, <i>supra</i>, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.</p>	<p>P. "Netlist" -- a structural description that includes a custom controller type hardware cell and all other hardware cells required to implement the application specific integrated circuit's operations and any necessary interconnections including the necessary control and data path information for connecting the hardware cells and the controller.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 12:31-35; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 58-60.</p> <p>Q. "generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit" -- producing a list of the needed hardware cells by eliminating any mapped hardware cells that are redundant or otherwise unnecessary and producing a custom controller type hardware cell for providing the needed control for those other hardware cells and</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<p><i>Intrinsic Evidence:</i> '432 patent: Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 61-62.</p> <p>R. "generating ...interconnection requirements therefor" -- producing the necessary structural control paths and data paths for the needed hardware cells and the custom controller.</p> <p><i>Intrinsic Evidence:</i> '432 patent: Abstract; Figs. 6 & 13-15; 1:17-37; 2:39-44; 3:23-25; 3:40-45; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585].</p> <p><i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 63-64.</p>
14. A process as defined in claim 13, including generating from the netlist the mask data required to	The process of claim 13, including producing from the netlist of hardware cells to be included in the designed ASIC mask data which can be directly used by a chip foundry in the fabrication of the ASIC.	S. "generating from the netlist the mask data required to produce an integrated circuit having the desired function" -- producing, from the structural netlist, the detailed layout level geometrical information required for

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
produce an integrated circuit having the desired function.	<p><u>Support</u></p> <p>'432 Patent: Figs. 1c, 2; column 1, lines 42-43; column 2, lines 44-49; column 3, line 68 to column 4, line 4; column 4, lines 44-46; column 5, lines 40-46.</p>	<p>manufacturing the set of photomasks that are used by the processes that directly manufacture the application specific integrated circuit.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: Abstract; Fig. 1c; 1:42-44; 1:54-58; 2:44-49; 4:44-46; 5:40-46; 14:4-6; 16:34-68 [Attachment 2, DEF012564-DEF012585]; '016 patent: 1:41-47; 1:57-61; 4:11-13 [Attachment 5, DEF017265-DEF017284].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶¶ 65-66.</p>
15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.	<p>The process of claim 13, including producing signal lines for carrying data to the hardware cells.</p> <p><u>Support</u></p> <p>'432 Patent: Figs. 11, 13; column 2, lines 39-40; column 3, lines 60-65.</p> <p>RCL000207-223.</p>	<p>T. "generating data paths for the selected integrated circuit hardware cells" -- producing the necessary structural descriptions of the data paths for the mapped hardware cells.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: Abstract; 2:39-40; 4:63-66; 5:6-12; 5:30-37; 6:29-31; 6:37-43; 6:50-53; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-3 [Attachment 2, DEF012564-DEF012585]; '016 patent: 2:21-25 [Attachment 5, DEF017265-017284].</p> <p><i>Extrinsic Evidence:</i></p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsis' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>Decl. Kowalski ¶¶ 67-69.</p> <p><i>Dictionaries, Treatises, Textbooks, etc.:</i> IEEE Standard Dictionary at 898 [Attachment 19, DEF085344-DEF085347].</p>
<p>16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.</p>	<p>The process of claim 15, wherein the step of producing signal lines for carrying data comprises applying rules, which are placed in computer memory, to produce the signal lines for carrying data to the hardware cells.</p> <p><u>Support</u></p> <p>'432 Patent: Column 4, lines 64-66.</p> <p>RCL000207-223.</p>	<p>U. "said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom" -- the generating step must be performed by at least an expert system having an inference engine for selectively applying a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to produce the necessary data paths for the mapped hardware cells.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: Abstract; 5:6-12; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-7 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 11 [Attachment 3, DEF011820-DEF012110]; '435 patent: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; An Overview of Logic Synthesis</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<p>Systems: at 170 [Attachment 7, DEF011962-DEF011968]; The CMU Design Automation System: at 75-77 [Attachment 8, DEF012045-DEF012052].</p> <p><i>Extrinsic Evidence:</i></p> <p>Decl. Kowalski ¶¶ 69-70.</p> <p>Same as set forth for phrases H, I, & J above.</p> <p><i>Dictionaries, Treatises, Textbooks, etc.</i></p> <p>Same as set forth for phrases H, I & J above.</p>
17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.	<p>The process of claim 16, including producing signal lines for carrying control signals to the hardware cells.</p> <p><u>Support</u></p> <p>'432 Patent: Figs. 11, 13; column 2, lines 40-42; column 3, lines 60-65.</p> <p>RCL000207-223.</p>	<p>V. "generating control paths for the selected integrated circuit hardware cells" --producing the necessary structural descriptions of the control paths for the selected hardware cells.</p> <p><i>Intrinsic Evidence:</i></p> <p>'432 patent: Abstract; Figs. 1b & 13-15; 1:17-37; 2:40-42; 3:59-65; 4:39-43; 4:63-65; 5:3-12; 5:30-36; 6:18-27; 11:49-51; 13:51-14:3; 16:34-65; 17:1-10 [Attachment 2, DEF012564-DEF012585]; '432 patent file history: April 1989 Amendment at 8 [Attachment 3, DEF011820-DEF012110]; '016 patent: 2:20-24 [Attachment 5, DEF017265-DEF017284].</p>

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		<i>Extrinsic Evidence:</i> Decl. Kowalski ¶¶ 71-72. <i>Dictionaries, Treatises, Textbooks, etc.:</i> IEEE Standard Dictionary at 898 [signal line], [Attachment 19, DEF085344-DEF085347].

EXHIBIT

35

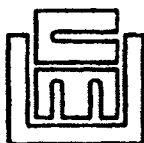
The VLSI Design Automation Assistant: A Knowledge-Based Expert System

Thaddeus Julius Kowalski

**SRC-CMU Center for Computer-Aided Design
Department of Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213**

Research Report No. CMUCAD-84-29

April 1984



Carnegie Mellon University

CARNEGIE-MELLON UNIVERSITY

**The VLSI Design Automation Assistant:
A Knowledge-Based Expert System**

**A DISSERTATION
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**

for the degree

**DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING**

by

Thaddeus Julius Kowalski

**Pittsburgh, Pennsylvania 15213
April 27, 1984**

**Copyright © 1984 by Thaddeus Julius Kowalski
All Rights Reserved**

Abstract of the Dissertation

The VLSI Design Automation Assistant:
A Knowledge-Based Expert System

by

Thaddeus Julius Kowalski

Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
April 27, 1984

VLSI design synthesis is a method for designing hardware that starts with an algorithmic description and uses interactive computer programs to create a finished design. This structured approach can decrease the time it takes to design a chip, automatically provide multi-level documentation for the finished design, and create reliable and testable designs. VLSI design synthesis is a difficult problem because the huge number of facts and implicit dynamic constraints do not lend themselves to a recipe-like solution. However, the Knowledge Based Expert System, KBES, approach provides a framework for organizing solutions to problems that are currently solved by experts using large amounts of domain-specific knowledge. Therefore, the goals of my research are to extract and codify the knowledge of expert designers for a better understanding of VLSI design-synthesis, to implement a KBES capable of creating efficient, testable and usable designs, and to determine the usefulness of the KBES technique for implementing design automation tools.

A series of acquisition interviews and an initial prototype system have been used to bootstrap a system that generates a technology-independent list of operators, registers, data paths and control signals from an algorithmic description. The Design Automation Assistant, DAA, has been used to design an IBM System 370 and was favorably evaluated by an IBM System/370 designer.

The advantages of using the KBES approach are the ease of validating the knowledge gathered from interviews with experts, the ease of incrementally adding to the knowledge base, the ability to query that knowledge during the design task, and the replacement of extensive backtracking by domain-specific knowledge techniques. The disadvantage is the difficulty of extracting knowledge from the experts.

This thesis adds to knowledge in the digital design synthesis domain by compiling and testing the set of rules used by expert designers, and to knowledge in the expert systems domain by providing another system for researchers to examine. This knowledge will also aid in the teaching of design by making explicit knowledge that is now passed on primarily through apprenticeship.

LIMITED DISTRIBUTION NOTICE

This report has been, or will be submitted for publication outside of CMU and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of CMU prior to publications is limited to peer communications and specific requests.

ACKNOWLEDGEMENT

First and foremost, my profound thanks go to my parents, Henry and Ann Kowalski. None of this could have been done without their indulgence, support, enthusiasm, and love. Thank you.

My sincere thanks goes to my adviser, Donald Thomas for his patience and support through many phone calls and meetings. I would like to thank Stephen Director, Allen Newell, John McDermott, and Daniel Siewiorek for their encouragement, support, and critical review of my research. Also, the thoughtful criticisms and suggestions of friends and colleagues have added greatly to this thesis and to my pleasure in writing it. In particular, David Ditzel, Mitch Marcus, Thomas Mitchell, Lisa Masakowski, and Myron Wish all read multiple versions with care. Marcia Derr, Charles Forgy, Karen Kekich, Michael McFarland, Sharon Murrel, Peg Schafer, and Sandra Frazansky have made helpful comments at various stages of the research. The many members of the CMU/DA community have provided many ideas and probing questions, especially David Gatenby, David Geiger, Charles Hitchcock, John Lertola, Matt Mathis, John Nestor, Jayanth Rajan, and Robert Walker. Invaluable assistance with the mechanics of my thesis was provided by the UNIX operating system, the TROFF text processing program, the writers workbench, and the OPSS knowledge-based expert-system writing tool.

I would like to thank Ken Chong, Claud Davis, David Ditzel, Michael Maul, Gil Mowery, Allen Ross, Clayton Schneider, Glen Williams, and Andrew Wilson for donating time to critique the various designs. Lastly, I would like to thank the INTEL and IBM corporations for providing designers to critique the designs, with special thanks to AT&T Bell Laboratories for computer resources and designer's critiques. This work was supported in part by an IBM fellowship and the National Science Foundation.

To William Shelley, my fourth grade teacher...

He made learning fun.

iii

If I Only Had A Brain

I could while away the hours
conferria' with the flow'rs
consaltin' with the rain
And my head, I'd be scratchin'
while my thoughts were busy hatchin'
If I Only Had A Brain.

I'd unravel ev'ry riddle
for any individie
in trouble or in pain
With the thoughts I'd be thinkin'
I could be another Lincoln,
If I Only Had A Brain.

Oh, I could tell you why the ocean's near the shore,
I could think of things I never think before
and then I'd sit and think some more.

I would not be just a stuffa'
my head all full of stuffa'
my heart all full of pain
And perhaps I'd deserve you
and be even worthy ev' you
If I Only Had A Brain.

"The Wizard of Oz"
Music: H. Arien
Words: E. Y. Harburg

Table of Contents

Chapter 1: INTRODUCTION	1
1.1 Motivation	2
1.2 CMU/DA Levels	3
1.3 Problem Statement	5
1.4 Approach	5
1.5 Related Research	6
1.6 Overview	9
Chapter 2: KNOWLEDGE-BASED EXPERT SYSTEMS	10
2.1 General Components of a Knowledge-Based Expert System	10
2.2 General Features of a Knowledge-Based Expert System	13
2.3 Example Knowledge-Based Expert Systems	13
2.4 KBES Summary	16
Chapter 3: DAA DEVELOPMENT	17
3.1 Methods to Acquire Expert Knowledge	17
3.2 The Case-Study Interviews	19
3.3 Prototype System	25
3.4 Knowledge Acquisition Interviews	29
3.5 Analysis of Knowledge	34
3.6 Development Summary	43
Chapter 4: DAA REPRESENTATIONS	46
4.1 Algorithmic Representation	46
4.2 Technology Database and Constraints	52
4.3 Technology-Independent Hardware Network	56
4.4 Bookkeeping Information	59
4.5 Representation Summary	60
Chapter 5: DAA KNOWLEDGE	62
5.1 Service Functions	64
5.2 Global Allocation	68
5.3 Value Trace Allocation	71
5.4 SCS Allocation	78
5.5 Estimators	83
5.6 Global Improvements	91
5.7 Knowledge Summary	95
Chapter 6: THE IBM SYSTEM/370 EXPERIMENT	98
6.1 The D370 Design	99
6.2 The μ 370 Design	105
6.3 The D370 and μ 370 Design Comparison	109
6.4 Summary of The System/370 Experiment	113
Chapter 7: CONCLUSION	114
REFERENCES	117
Appendix A: WORKING-MEMORY VT	124

Appendix B: WORKING-MEMORY USER PARAMETERS	126
Appendix C: WORKING-MEMORY SCS	130
Appendix D: THE SYSTEM/370 CRITIQUE	132
D.1 The Critique	132
D.2 The ICC Design	136
D.3 Back To Critique	147
D.4 Summary	159

List of Figures

Figure 1. MCS6502 — 8 BIT MUX DATA PATHS	32
Figure 2. MCS6502 — 8 BIT BUS DATA PATHS	33
Figure 3. MODULE-TO-MUX-AND-BUS-WITH-BUS-TO-MUX	36
Figure 4. ASSIGN-INC	37
Figure 5. BIND-ARITHMETIC-OPERATORS-SAME-SIZE	37
Figure 6. ALLOCATION	38
Figure 7. NEXT	38
Figure 8. DEFAULT-PLUS	39
Figure 9. DEFAULT-FOLD-ARITHMETIC	39
Figure 10. CLEAN-OUTNODES	40
Figure 11. DECLARED-VARIABLE-ALLOCATION-DONE	40
Figure 12. TRIM-ADD-SUB-MUL-OUTPUT-ZERO-NIL-TRIM	41
Figure 13. TRIM-OUTPUT	41
Figure 14. NEXT-LIST-1	42
Figure 15. END-LIST-1	42
Figure 16. FOLD-REGISTER-PROX	43
Figure 17. NOT-STABLE-REGISTERS	43
Figure 18. SAMPLE ISPS DESCRIPTION	47
Figure 19. SAMPLE VT	48
Figure 20. SAMPLE STRUCTURAL SPECIFICATION	56
Figure 21. SAMPLE CONTROL SPECIFICATION	57
Figure 22. DAA SUBTASKS	62
Figure 23. MAKE-LINK	67
Figure 24. GLOBAL ALLOCATION	68
Figure 25. STATE OF DESIGN AFTER GLOBAL ALLOCATION	69
Figure 26. FIND-VTBODIES-FOR-REGISTERS	70
Figure 27. VT ALLOCATION	71
Figure 28. STATE OF DESIGN AFTER VALUE TRACE ALLOCATION	72

Figure 29. FIND-OUTNODES-FOR-TEMPORARIES	75
Figure 30. SCS ALLOCATION	79
Figure 31. STATE OF DESIGN AFTER SCS ALLOCATION	80
Figure 32. FOLD-NO-OUTPUT-REGISTER	81
Figure 33. GLOBAL IMPROVEMENTS	91
Figure 34. STATE OF DESIGN AFTER GLOBAL IMPROVEMENTS	92
Figure 35. CONVERT-MUX-INPUTS-TO-BUS	93
Figure 36. AN EXAMPLE BUS ALLOCATION	94
Figure 37. THE D370 DESIGN – PART 1	100
Figure 38. THE D370 DESIGN – PART 2	101
Figure 39. THE μ 370 DESIGN	106

List of Tables

Table 1. A DECADE OF SYSTEMS	14
Table 2. SUMMARY OF INTERVIEWS – PART 1	20
Table 3. SUMMARY OF INTERVIEWS – PART 2	21
Table 4. MCS6502 – THREE DESIGNS	31
Table 5. RULES BY KNOWLEDGE TYPE	35
Table 6. VTBODY AND OUTNODE	50
Table 7. OPERATOR, BRANCHES, LISTS AND TREES	51
Table 8. USER PARAMETERS	63
Table 9. DEFAULT DB-OPERATOR – PART 1	54
Table 10. DEFAULT DB-OPERATOR – PART 2	55
Table 11. DEFAULT FOLD AND DELAY	56
Table 12. HARDWARE	58
Table 13. MISCELLANEOUS	60
Table 14. RULES BY FUNCTION AND KNOWLEDGE TYPE	63
Table 15. MEMORY ARRAYS IN THE D370 DESIGN	103
Table 16. REGISTERS IN THE D370 DESIGN	104
Table 17. IBM SYSTEM/370 – DESIGN DIFFERENCES	110

Chapter 1: INTRODUCTION

Recent advances in integrated circuit fabrication technology have allowed larger and more complex designs to form complete systems¹ on single VLSI chips. These chips use one-micron to five-micron features to achieve complexities equivalent to 100,000 to 250,000 transistors. This level of design complexity has created a combinatorial explosion of details — a major limitation in realizing cost-effective, low-volume, special-purpose VLSI systems. To overcome this limitation, design tools and methodologies capable of automating more of the digital synthesis process must be built.

We have been developing just such synthesis tools^{2,3} in the Carnegie-Mellon University design automation, CMU/DA, community. These tools help the designer develop the algorithmic description of the system and interactively add the details required to produce a finished design. This structured approach can decrease the time it takes to design a chip, automatically provide multi-level documentation for the finished design, and create reliable and testable designs.

This thesis focuses on the synthesis, or allocation, of the implementation-design space as it advances from an algorithmic description of a VLSI system to a list of technology-independent registers, operators, data paths and control signals. Our approach is aimed at aiding the designer by producing data paths and control sequences that implement the algorithmic system description within supplied constraints. Thus, the designer can consider many alternatives before deciding on a final design.

This task has inspired a variety of approaches, ranging from the most simplistic backtracking methods through the most complicated constraint propagation methods.^{4,5,6,7,8} Owing to the complexity of design synthesis, simplistic backtracking

schemes consume large amounts of CPU time, and the constraint propagation method is too cumbersome for large designs. Because of the combinatorial explosion of details and implicit dynamic constraints involved in choosing an implementation, this problem does not lend itself to these algorithmic solutions. An alternate approach to design synthesis uses a large amount of design knowledge to eliminate backtracking; whenever possible, the focus is on specific design details and constraints. Artificial intelligence researchers have called systems developed under this heuristic approach knowledge-based expert systems, KBESs.⁹ This chapter motivates the research, provides background on the CMU/DA system, lists the related research in the area, states the problem, and provides a road map of the rest of the thesis.

1.1 Motivation

If Moore's law¹⁰ continues to hold, within this decade the size of the smallest VLSI feature will be reduced ten times. This increasing potential to fabricate more complex systems will cause at least a linear increase in information that must be managed. From information management studies of large software projects,¹¹ it can be shown that the time to design and implement future VLSI systems will be considerably more than ten times that required at present, which is prohibitive for low-volume special-purpose VLSI systems. It can be shown further that as the cost of designing special-purpose VLSI systems decreases, the demand for expert designers will increase. Thus, design methods should be developed to deal effectively with the magnitude and complexity of VLSI design. Such methods would increase the potential productivity of designers, while also making it possible for more people to design systems that are both testable and reliable. These are areas even good designers often forget, but can mean the difference between working and marginal designs.

Many people are working on problems related to computer-aided design, or CAD, of VLSI systems. For example, if we look in the proceedings of the Twentieth Design Automation Conference, we see topics of hardware description languages, testing, simulation, layout and placement, PLA minimization, and synthesis, to name a few. Although synthesis was considered to be strictly in the realm of the creative designer, automatic and computer-aided synthesis programs are now being developed for many levels of VLSI system design. In general, the quality of the designs produced by automatic synthesis programs is not adequate for complete automation of the design process for production use. However, these programs are beginning to find use as an interactive aid during the design process.^{2,12,13,14} Toward this end the development of synthesis tools to aid in the creative design process has become an important area of research.

1.2 CMU/DA Levels

At CMU we look at synthesis as the creation of a detailed representation from an abstract representation. The VLSI design synthesis task can be decomposed into several subtasks, each providing an increasing level of detail from the abstract representation. This section describes the four levels of increasing detail used in the CMU/DA system.²

1.2.1 The algorithmic level. The first level is an algorithmic description of the design. At this level of detail the high-level intent of hardware can be understood and simulated¹⁵ regardless of the target design style (for example pipeline, multiplexer, microprocessor, or bus) or technology. This level is represented in the instruction set processor language, ISPS,¹⁶ and a value trace data and control flow description language, VT.^{17,18} ISPS is a programming language similar to ALGOL, while VT is an extraction of the data flow and control flow information present in the designer's ISPS description. VT is easy for computer

programs to manipulate and is felt to be less sensitive to different styles of writing the same algorithmic description.

1.2.2 The technology-independent-hardware-network level. The second level is a technology-independent-hardware-network description of the design. At this level of detail the functionality and connectivity of the hardware can be understood regardless of the target design technology (TTL, ECL, NMOS, or CMOS). This level is represented in the technology-independent-hardware-network language, SCS,¹⁹ which describes technology-independent registers, operators, data paths, and control signals.[†] Data-memory allocation and control allocation are defined as the creation of this level from the algorithmic level while applying classical compiler optimizations²¹ and design styles.²²

1.2.3 The technology-dependent-hardware-network level. The third level is a technology-dependent-hardware-network description of the design. At this level of detail the logic and circuits of the hardware can be understood and simulated²³ regardless of physical placement. This level is represented in the technology-dependent-hardware-network language, DIF,²⁰ which describes feature assignment detail. Feature binding is the creation of this level from the technology-independent level by selection of registers, operators, data paths and control signals that match available feature data-base entries.^{24, 25, 26}

1.2.4 The fabrication-dependent-hardware-network level. The fourth level is a fabrication-dependent-hardware-network description of the design. At this level of detail the physical placement of hardware can be understood and simulated.²⁷ This level is represented in a fabrication-dependent-hardware-network language, which describes feature

[†] SCS is soon to be replaced by DIF.²⁰

placement assignment detail. Layout is the creation of this level from the technology-dependent level using geometric information to guide routing and placement of features.

1.3 Problem Statement

Now that we understand how the CMU/DA system divides the task into synthesis levels, let us examine how expert VLSI designers make the transition from the algorithmic description to a hardware implementation. This thesis examines how expert VLSI designers choose a hardware implementation for MOS-microcomputers and whether a KBES can mimic their results. The goals of the research are to extract, codify and test the knowledge of expert designers for a better understanding of VLSI design-synthesis, to implement a KBES capable of creating efficient, testable and usable designs, and to determine the usefulness of the KBES technique for implementing design automation tools. This adds knowledge in the digital design synthesis domain by compiling and testing the set of rules used by expert designers, and to knowledge in the expert system domain by providing another system for researchers to examine. This knowledge will also aid in the teaching of design by making explicit knowledge that is now passed on primarily through apprenticeship.

1.4 Approach

Through a series of detailed structured interviews, the knowledge that expert VLSI designers use to go from the algorithmic level to the technology-independent hardware-network level has been extracted. The codified knowledge has been tested for completeness and correctness by implementing an interactive system, DAA, that adds register, operator, data path and control signal detail to the VT description of hardware to form the SCS description. DAA is implemented as a production system using the OPS3²³ KBES writing

system. This KBES tool is based on the premise that humans solve problems by recognizing familiar sub-problems and apply past solutions. This domain is appropriate for a KBES because there are human experts available whose knowledge has been gained through experience and who can teach this knowledge through apprenticeship. Furthermore, the knowledge required to do the task is extensive and requires the type of organization provided by the KBES approach. The advantages of using the KBES approach are the ease of validating the knowledge gathered from interviews with experts, the ease of incrementally adding to the knowledge base, the ability to query that knowledge during the design task, and the replacement of extensive backtracking by domain-specific knowledge techniques. The disadvantage is the difficulty of extracting knowledge from the experts.

The DAA system has been used to design many computers including the MOS Technology Incorporated MCS6502 and the IBM System/370. The design and many redesigns of the MCS6502 provided stimulus for critiquing the implementation design knowledge contained in DAA. A design of each of the small ISPS descriptions maintained at CMU (RK11, HP21MX, FB, 18080, MOD91, 1802, AM2903, H316, VIDEO, TI1200, PDFTTY, SCF3, PDM4, FP, AM2910, AM2901, ELEV, CHANGE, PQ, MINI, MINIS, AM2909, FM07, AM2902, MARK1) showed that the system would produce a functionally correct design for a large number of test cases. Finally, the knowledge in DAA was tested for generality and robustness by designing a much larger processor, with a completely different instruction set, than DAA had ever seen before, and critiquing this design with a designer not used to develop the knowledge base.

1.5 Related Research

The work done thus far in digital design synthesis is reminiscent of the early chess

playing programs.²⁹ It is easy to get a computer to play a legal game of chess, but it is exceeding difficult to get it to play a good game of chess. The CMU/DA project, like the legal game of chess, gets a computer to generate a working piece of hardware from an algorithmic description.³⁰ Previous attempts at hardware allocators have either produced designs of unacceptable quality, or have been too expensive computationally even for small designs. This section discusses the shortcomings of the current hardware allocators.

At first glance the problem of hardware allocation appears to be a straightforward extraction of registers, operators, and connections from an algorithmic description, but this overlooks the use of multiple design styles and important constraints like cost, size, power and speed. This classical compiler approach was employed by one allocator⁶ which used multiplexer-style data flow; it was found to produce designs that were considerably worse than those made by humans.³¹

The MIMOLA Software System, MSS,⁴ is an interactive computer-aided design system that takes into consideration hardware usage and limits on the amount of hardware. The MSS allocator starts with a maximally parallel algorithmic description and serializes it until the limits on the amount of hardware can be met. It does this using a single design style and user-provided hardware restrictions. The MSS allocator makes tradeoffs one line at a time within the algorithmic description. Thus, it is limited to local optimizations.

Further research has taken a constraint language approach to the allocation problem,⁵ which uses a mixed-integer linear programming solution technique. Though this approach does not have the previous shortcomings, it allows only trivial problems to be solved because its solution method has non-polynomial hard complexity characteristics.³²

A graph theoretic approach to digital data path synthesis has been developed in a

program called Facet.⁶ It first develops a maximally parallel dependency representation. Then it allocates the minimum number of storage elements, data operators, and interconnections units for each disjoint cluster, or clique, of algorithmic operators and values in the design. This approach does not consider testability, ease of layout, or the flexibility of mixing alternate design styles.

Another algorithmic approach has been developed in a program called EMUCS.⁷ The algorithm aims at an implementation at minimum "cost" for any quantitative parameter such as power or chip area. The algorithm allocates, in a step by step fashion, processors, registers, and multiplexer and bus transfer paths. The synthesis algorithm, as proposed by McFarland,³³ is iterative in nature. It first analyzes the existing intermediate data path to decide what hardware would potentially be the least costly to bind, not according to the lowest cost during this step, but because it might minimize the costs in the next iteration. It then binds that one element, changing the data path as necessary and iterates, reanalyzing and binding, until all elements have been bound and the final data path created. The designs it creates are good locally, but could be improved with a global view including layout, testability and reliability.

Palladio³⁴ is a circuit design environment for experimenting with methodologies and knowledge-based expert-system design aids. Its framework is based on the premise that circuit designers need an integrated design environment ranging from implementation simulators to layout generators. Their goal is to provide the means for explicitly representing, constructing and testing design tools and languages. This environment integrates both design tools and design specification languages and provides the conceptual framework required by such integration. It is an attempt to explore how to build circuit design environments rather than an automated synthesis program.

1.6 Overview

This chapter has discussed the growing complexity of the VLSI design synthesis task brought on by the decrease in cost and increase in ability to fabricate VLSI systems, and the limited number of expert VLSI designers. It has overviewed the CMU solution to the problem, specifically highlighting the area of implementation synthesis and the various attempts to solve this problem. It discussed an alternate approach to design synthesis, which uses a large amount of design knowledge to eliminate backtracking; whenever possible, the focus is on specific design details and constraints. This approach creates flexible and testable designs that are sensitive to technology, cost, and layout considerations.

Chapter 2, which provides an overview of the knowledge-based expert-system technique, can be skipped by readers familiar with this approach. Chapter 3 discusses the development cycle of DAA, including the interviews, which led to a prototype system, and the further interviews that have developed the DAA system. Chapters 4 and 5 provide insight into how VLSI systems can be designed by discussing the data representations and the task knowledge used by DAA. This knowledge is tested in Chapter 6 with an IBM System/370 design and critiqued by an expert designer at IBM. Finally Chapter 7 summarizes the thesis and discusses possible future research. In summary, this thesis focuses on the synthesis of the implementation-design space and provides insight into how a KBES approach can be used to design cost-effective, low-volume, special-purpose VLSI systems.

Chapter 2: KNOWLEDGE-BASED EXPERT SYSTEMS

During the past decade KBESs have been developed by researchers in artificial intelligence to help solve problems whose structures do not lend themselves to recipe-like solutions. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. This tool is based on the premise that humans solve problems by recognizing one of many familiar patterns in the current situation and applying the appropriate actions when this pattern occurs. Their recognition of the pattern is not based on the complete current situation, nor is it recognized with absolute certainty. The presence and absence of patterns can be used to help establish and rule out actions for a given situation.

Researchers have developed many KBESs whose features³⁵ have matured and grown into usable engineering tools.^{36,37,38} These systems exploit special knowledge to solve difficult problems in many domains. Examples are: DENDRAL, mass-spectrum analysis; MYCIN, medical diagnosis; PROSPECTOR, mineral exploration; R1, VAX computer configuration; to name only a few. This chapter introduces general components of KBESs, describes their general features, describes two implementations, and provides references to several other systems. This chapter introduces the general features of KBESs, while Chapter 3 discusses how the KBES approach has been used to develop an intelligent CAD tool, DAA.

2.1 General Components of a Knowledge-Based Expert System

The problem domains and features of the existing KBESs differ widely, but many have

three components is common: a working memory, a rule memory, and a rule interpreter.

2.1.1 The working-memory component. The working memory is a collection of elements that describe the current situation. The elements resemble the data structures in conventional programming languages:

```
struct foo {
    <attribute 1> = <value 1>;
    ~
    <attribute n> = <value n>;
};
```

Some systems also represent goals and sub-goals as named attribute-value pairs in working memory.

2.1.2 The rule memory component. The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:

```
IF:
    <antecedent 1>
    ~
    <antecedent n>

THEN:
    <consequence 1>
    ~
    <consequence m>
```

The rule memory is a collection of knowledge *chunks* about a particular problem domain. Most rule-based systems contain hundreds of rules that have been *painfully* extracted from months of interviewing experts. Acquiring knowledge from experts is difficult because although they are skillful at doing the task, they are not effective at explaining precisely how they do the task. To give a feel for the number of rules: MYCIN has about four-hundred fifty rules, R1 has about eight-hundred fifty rules (the new version

XCON has three-thousand three-hundred rules),⁷⁹ and PROSPECTOR has about one-thousand six hundred rules. Systems like MYCIN have added debugged rules at an average rate of about two per week. This in no way accounts for the hundreds of rules that came and went in the debugging of the rule memory.

2.1.3 The rule interpreter component. The rule interpreter pattern matches the working-memory elements against the rule memory to decide what rules apply to the given situation. Some tool-kits allow a degree of certainty to be associated with each consequence that suggests the degree to which the consequence follows from the antecedents. Others allow a pair of certainties to be associated with each consequence that suggests both how sufficient the presence of the antecedents are for establishing the consequence and how sufficient the absence of the antecedents are for not establishing the consequence. Still others apply the consequences with absolute certainty if the antecedents are present in working memory. The selection process of rules can be data driven, goal driven, or a combination of data and goal driven.

A data-driven selection process looks through the rule memory for a rule whose antecedents match elements in the working memory. This is also called forward-chaining or antecedent reasoning. The consequences of the rule are applied, and the process is repeated until no more rules apply or until a rule explicitly stops the process.

A goal-driven selection process looks through the rule memory for a rule whose consequences can achieve the current goal. This is also called backward-chaining or consequent reasoning. If the antecedents of this rule match the elements in working memory, then the consequences are applied and the goal is satisfied. If the antecedents of this rule are not all present in working memory, then the missing antecedent replaces the

current goal and the process is repeated until either all the sub-goals are satisfied or no more rules are applicable. If no more rules are applicable, the user can be queried for missing information to place in working memory. This backward-chaining reason also facilitates explanations of how the system had reached a particular conclusion and why it needed certain pieces of knowledge.

2.2 General Features of a Knowledge-Based Expert System

Separating expert knowledge from the reasoning mechanism implies several general features common to knowledge-based expert systems. The knowledge engineer can incrementally add new rules and refine old ones because the rules are chunks of domain knowledge and have minimal interaction with other rules in the rule memory. The rule interpreter can be queried about why it needs an additional piece of information and how it solved a problem by describing the goals and the rules it has used to solve a problem. A rule interpreter can be developed for aiding the acquisition of knowledge,⁴⁰ by checking its own rule memory for oddity, consistency and omitted areas. A rule interpreter can also be developed for aiding the instruction of knowledge,⁴¹ by trying to determine what knowledge students seem to possess and how that knowledge corresponds to its rule memory. Finally, the same rule interpreter may be used by many rule memories and working memories to develop KBESs for different problem domains.

2.3 Example Knowledge-Based Expert Systems

During the past decade many KBESs have been developed for such purposes as interpretation, diagnosis, monitoring, prediction, planning and design. This section discusses two examples — a medical diagnostic system, MYCIN, and a system used to configure VAX computers, R1, and provides references to further readings in Table 1. The commentary

System	Domain
INTERLIST ⁴² MYCIN ⁴³ PUFF ³⁷ GUIDON ⁴⁴ VM ⁴⁵	diagnose in medicine diagnose in medicine diagnose in medicine computer-aided instruction in medicine measurement interpretation in medicine
SACON ⁴⁶	diagnose in structural engineering
PROSPECTOR ⁴⁷	diagnose in geology
DENDRAL ⁴⁸ SECS ⁴⁹ SYNCHM ⁵⁰	mass-spectrum analysis in chemistry organic chemistry chemistry
SADD ⁵¹ EL ⁵² PALLADIO ⁵⁴ SOPHIE ⁵³	electronics circuit analysis VLSI design computer aided instruction in electronics
MOLGEN ⁵⁴	problem solving and planning in genetics
NEWTON ⁵⁵	problem solving and planning in mechanics
PECOG ⁵⁶	problem solving and planning in programming
DART ⁵⁷ NLS ⁵⁸ XSEL ⁵⁹	diagnose in computer faults configuring VAX computers computer sales person assistant
CONCHIE ⁶⁰	knowledge acquisition
TEIRESIAS ⁶⁰	knowledge acquisition for MYCIN systems
HEARSAY-II ⁶¹	speech recognition
AGE ⁶² EXPERT ⁶³ EXPERT ⁶⁴ HEARSAY-II ⁶⁵ KAS ⁶⁶ META-DENDRAL ⁶⁷ NONMYCIN ⁶⁸ AMORD ⁶⁹ OPS ⁷⁰ OPS ⁷¹ ROSIE ⁷¹	blackboard model tool-kit MYCIN tool-kit diagnose in medicine HEARSAY-II tool-kit PROSPECTOR tool-kit DENDRAL tool-kit computer aided instruction in MYCIN as EL tool-kit as OPS family tool-kit as OPS family tool-kit a RITA tool-kit

Table 1. A DECADE OF SYSTEMS

provides a description of the task and a discussion of the rule interpreter with its current states.

2.3.1 The MYCIN example. MYCIN is an interactive program for medical diagnosis. It produces diagnoses of infectious diseases, particularly blood infections and meningitis infections, and advises the physician on antibiotic therapies for treating those infectious diseases. During the consultation, which is conducted in a stylized form of English, the physician is asked only for patient history and laboratory test results. The physician may ask for an explanation of the diagnosis and the reason a laboratory test result is required.

MYCIN uses a goal-driven rule interpreter to scan a rule memory of about five hundred rules covering meningitis and blood infections. Each rule has a certainty factor that suggests the strength of the association or degree of confidence between the antecedents of the rules and the consequences of the rules. MYCIN has equaled the performance of nationally recognized experts in diagnosing and recommending therapy for meningitis and blood infections. However, it is not clinically used because the human factors of its interface do not save the doctors any time and its knowledge is limited to these two domains.

2.3.2 The RI example. RI is a program for configuring VAX computers. It organizes the customer selected components by location relative to other components or to the rooms in which the system will be housed, and it specifies the cabling required to connect pairs of components. This configurator is given only the set of components selected by a customer and produces the list of missing components, components in each CPU cabinet, components in each UNIBUS box, distribution panels in each cabinet, the floor layout, and cabling requirements.

R1 uses a data-driven rule interpreter to scan a rule memory of about eight hundred rules covering component configuration. These rules are written in the OPS5 programming language and use a direct association between antecedents and consequences. It takes about seventy five seconds on a VAX 11/780 to configure a typical order of about 90 components. R1 has equaled the performance of Digital Equipment Corporation experts in configuring VAX computers and is currently being used in a production mode.

2.4 KBES Summary

During the past decade many KBESs have been developed by researchers in artificial intelligence that assist experts in solving real-world problems such as interpretation, diagnosis, monitoring, prediction, planning and design. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. Development of KBESs is aided by the separation of working memory, rule memory, and rule interpreter, which enables how and why questions to be asked. Researchers have developed many KBESs whose features have matured and grown into usable engineering tools. Most importantly, this tool is appropriate to the design domain because there are human experts available whose knowledge has been gained through experience and who can teach this knowledge through apprenticeship. Furthermore, the knowledge required to do the task is extensive and requires the type of organization provided by the KBES approach.

Chapter 3: DAA DEVELOPMENT

KBESs are generally developed in several stages. First, "book knowledge" of the problem is codified as a set of situation-action rules; interviews with experts then fill in knowledge gaps and refine current knowledge. Then, many example problems are given to the KBES, and experts closely examine and validate the results. Often, errors are found through the examples, and new rules are added to the system to correct the error situations.

This iterative process is necessary because experts are often unaware of exactly how they go about designing a chip and are inexperienced at articulating the procedure. Furthermore, the knowledge base is not an exact codification of the expert's knowledge, but a compilation of what is understood by the knowledge engineer.

This section discusses the development cycle of DAA, starting from the case-study interviews that led to the prototype system, followed by the acquisition interviews that have developed the DAA system, and concluding with an analysis of the acquired knowledge.

3.1 Methods to Acquire Expert Knowledge

The first step to extract knowledge that expert VLSI designers use to do allocation is to examine elicitation methods used by professionals. Extracting knowledge from experts is difficult because although they are experts at doing the task, they are likely to be novices at explaining precisely how they do the task. Though acquiring expert knowledge is a difficult problem, there are many elicitation procedures in psychology and cognitive science. A spectrum of detail, from broad-based to specific, can be elicited by using techniques that are unstructured, interviewer-structured, or interviewee-structured, depending on purpose and taste. This section discusses two elicitation procedures and a characterization of designer expertise.

A case-study is an interviewee-structured face-to-face session in which the expert is asked to explain how a task is done. This approach gives information about a few of the problems the expert has encountered, but not the complete set of possible problems. This may be because an expert "knows" what to do by recognizing a familiar pattern in the current situation and remembering what actions to take for that situation. Thus, the expert must be presented with a pattern to remember what to do. Acquiring knowledge from several experts by this approach yields a broad base of isolated pieces of information, which is a good method for initial knowledge acquisition.

An acquisition interview is an interviewer-structured face-to-face session presenting a specific problem and its solution, and asking the expert to point out where and why the solution went wrong. This approach can uncover the unspoken steps for doing a task. Acquiring knowledge from several experts by this approach fills in the gaps from previous knowledge acquisition attempts and provides a good method for verifying and testing the acquired knowledge.

Along with the many elicitation methods, there are three possible populations of designers to interview, each of which has its own strong and weak points:

- An inexperienced designer may not have a well formed method of doing design, but may be better at explaining the text book designs.
- A designer with a moderate amount of experience may be starting to form design methods based on what has worked in previous designs, but may have forgotten some of the reasons for the original decisions.
- An expert designer has the best formed design methods, but may only be able to explain the decisions when pushed for the reasoning.

If DAA is to model expert behavior, it is important for the acquisition interviews to use experts. On the other hand, for the case-study method, it would be informative to use designers from all three ranges of experience. Thus, all types of designers were used in the case-study interviews, while only expert designers were used for the acquisition interviews.

3.2 The Case-Study Interviews

After gathering current book knowledge about synthesis of the implementation design space,^{6,4,5} we taped interviews with four designers and summarized the interviews. This section overviews those interviews, while the next section discusses the design of the prototype system.

We interviewed four designers of varied experience: one was a novice, two were moderately experienced, and one was an expert. The interviews, which lasted about an hour each, started with a determination of the designer's background, including years of experience, logic families used, and designs created. Most of the time was spent discussing the design process, with some time given to a discussion of the DAA system. The designers discussed the global picture, partitioning, selection and allocation tasks, each with sub-tasks or attributes. Our interview method was designed to allow the interviewees as much freedom as possible in generating ideas; we emphasized such questions as:

"What do you do next?"
and
"Could you elaborate?"

Using this interview method the designers found it hard to be specific about tasks and sub-tasks. However, by taping the interviews our interaction and involvement with the designers could encourage them to be specific.

20

Table 2. SUMMARY OF INTERVIEWS – PART I

Item	Interview 1	Interview 2	Interview 3	Interview 4
Background				
Level	novice	moderate	expert	moderate
TTL & ECL	yes	yes	yes	yes
NMOS	no	yes	yes	yes
CMOS & PMOS	no	no	yes	no
Industrial years	3	4	12	4
Total years	7	9	12	7
Global Picture				
Inputs	yes	yes	yes	yes
Outputs	yes	yes	yes	yes
Constraints	yes	yes	yes	yes
Functionality	yes	yes	yes	yes
Feasibility	yes	yes	yes	yes
Technology Independence	yes	no	yes	no
Partitioning				
Functions	yes	yes		
Communication	yes	yes	yes	yes
Style	yes			
Nodes			yes	
Ordering				
Constraints	1	1	1	1
Output, Input	2	2		
Regular Structures			2	2
Initial Style				
Parallel	yes	yes		yes
Old Designs	yes	yes		yes
Serial			yes	
Allocation				
Clock Phases	1		2	
Operators	2	3		
Registers	3	2	3	
Data Paths	4	1	1	1
Control Logic	5	4	4	2

The first interview served as an initial exploration of the interview technique. It was difficult to get the designer to talk about the specifics of the design process without adding a

Table 3. SUMMARY OF INTERVIEWS – PART 2

Item	Interview 1	Interview 2	Interview 3	Interview 4
Iterating				
Constraint Violations	yes	yes	yes	yes
Global Improvements	yes			yes
Technology Increase	yes	yes	yes	yes
Functionality Decrease		yes	yes	yes
Constraints				
Speed	yes	yes	yes	yes
Area		yes	yes	yes
Power			yes	yes
Schedule			yes	
Cost	yes	yes	yes	
Drive	yes			yes
Width	yes	yes		
Bottom-Up				
After Global Picture	yes	yes	yes	yes

bias. Three more designers were interviewed in the next few days.

As expected, the case-study interviews gave information about a few of the problems the experts had encountered, but not the complete set of possible problems. When they did discuss details, they spoke about recognizing a situation and taking the appropriate action. The case-study method gave background knowledge that provided a framework for the other knowledge acquisition methods, but was insufficient for building the complete DAA system.

Tapes of the case-study interviews were replayed several times to construct a two-level classification of responses by each person. Each designer was given the information from his interview and asked to review it for accuracy and omissions. Tables 2 and 3 show the classifications, while the following sections explain the entries. A *yes* entry in the table shows the designer mentioned the item and said he *did* do it. Even though many entries are marked *yes*, they do not mean each designer did the same thing. A *no* entry in the table

shows the designer mentioned the item and said he *didn't* do it. A missing table entry shows the designer never mentioned the item. A *numeric* entry shows the designer mentioned the item and ascribed an order to this item with relation to other items he mentioned.

3.2.1 *The background of the designers.* Four designers^{72,73,74,75} of varied experience were interviewed: one was a novice, two were moderately experienced and one was an expert. The ratings were established by the amount of experience each designer had with MSI and LSI design techniques.

Designer-1 is a novice designer and home-computer hobbyist. He has been designing with TTL and microprocessor-based systems for about seven years. During the last three years he has worked for several small companies as a design engineer. He has designed memory systems, terminal interfaces, cassette tape recorder interfaces, and modem interfaces.

Designer-2 is a moderately experienced designer and home-computer hobbyist. He has been designing with TTL, ECL, and a little NMOS for nine years. During the last four years he has been working in industry and pursuing his doctorate. He has designed a bus controller for a disk drive and a Digital Equipment Corporation Q-Bus for a Motorola 68000 microprocessor.

Designer-3 is an expert industrial designer. He has been designing with TTL, ECL, NMOS, and CMOS for twelve years. He has designed a programmable controller, a four-bit microprocessor and is now managing a team designing a new microprocessor.

Designer-4 is a moderately experienced industrial designer. He has been designing with TTL, ECL, and a little NMOS for seven years, the last four years in industry. He has

designed a high-resolution bit-mapped display and the registers and ALU of a 32-bit microprocessor.

3.2.2 Global picture of the design. The designers began with a high-level overview of the hardware, which listed inputs and outputs to the outside world, the functions the hardware should provide, and general constraints. They discussed design feasibility with considerations of the target technology. Two designers internalized the global picture in functions allowed by the target technology. Designer-4 said, "There is always a right logic family for a design problem."

3.2.3 Partitioning a design problem. The designers partitioned the global picture into smaller units to be dealt with separately. They emphasized minimizing connections among blocks. One designer also partitioned by groups that operated as parallel or serial units. Two designers put functions/operators of similar types together. One designer partitioned by common multiplexer or communication bus.

3.2.4 Ordering the partitions within a design. Once the design was partitioned, the partitions were chosen for allocation in a decreasing order of difficulty or degree of constraint. Designers-1,2,3 reasoned that if the most difficult part could be designed, the rest of the design was feasible. Two designers worked in a grand sweep fashion: from output to input and input to output, from left to right, or from top to bottom. The other two designers continued by selecting regular structures next.

3.2.5 Initial style. Once a partition was selected for allocation, it was carried out either in parallel or in series. Designers-1,2,4 said, "A parallel design made thinking of the control logic much simpler," while designer-3 said, "A serial design minimized the design area." The constraints of the parallel design were examined for size violations to determine the

parts to be serialized by adding data paths, registers, and control logic to the initial parallel design. The constraints of the serial design were examined for speed violations to determine the parts to be reimplemented in parallel. If the designers recognized a part of the design as similar to a part of a previous design, they used what they knew had worked in the past.[†]

3.2.6 Allocation. Within each partition, designers allocated clock phases, operators, registers, data paths, and control logic. The order was interesting because once registers and data paths were allocated, they were not changed. Designer-2 said, "The control was changed after the data paths because it was the hardest thing to think about and because it depended on a constant structure of data-path elements." Designer-3 chose registers by what values had to be latched between clock phases/cycles. Designers-2,3 picked data paths by a general style of multiplexer or bus, depending on the expected amount of traffic between the two points.

3.2.7 Iterating through a design's lifetime. The designers described the iteration process as a step-by-step refinement to meet violated constraints. They looked for a technology change to meet a constraint before making a design change. This could be as simple as finding a new chip in the TTL data book or as complicated as a design-rule shrink. Three designers described giving up functionality to meet a constraint. Other design changes consisted of global improvements not recognized until the design neared completion. This suggests that the general choice of partitions and the initial design style selections approached optimum. This is an important point in the later development of the DAA system because designers do not seem to see backtracking in their designs. The designers were willing to make local

[†] I find this recognition of past designs is like the opening book moves for many of the chess playing computer programs.

iterations and improvements to clean up the design, but once they allocated a piece of hardware they did not retract it.

3.2.8 Use of design constraints. The whole design process seems to be a large multi-variable constraint problem. Designer-4 summed it up best by saying,

"An engineer's training teaches when constraints can be swept under the rug."

The relative importance of constraints is application dependent. The designers mentioned the constraints of speed, area, power, schedule, cost, drive capabilities, and bit width.

3.2.9 Top down to bottom up designs. The designers started with a top down approach, and then chose a bottom up approach for designing the most constrained partition. The switch to a bottom up approach seemed to occur after the global picture was completed. Designers-1,3 iterated using both the top down and bottom up paradigm. Designer-4 proceeded mostly top down, possibly because of his hierarchically oriented design tool, DRAW,⁷⁶ and software training. The other designer proceeded without a particular order.

3.3 Prototype System

Even though many details were missing, enough book knowledge had been gathered to put together a prototype version of the DAA system using the OPS5²⁹ KBES writing system. The initial knowledge in the system was codified from the algorithms of the current CMU/DA allocator⁶ and the interviews discussed above. Throughout the development of the DAA system the experts interacted with the knowledge engineer, not the rules themselves. While the DAA system was far from perfect at this point, it stimulated further elicitation sessions with expert designers. This section discusses the flow of control in the prototype system and how the KBES approach formulates the problem. Chapters 4 and 5

present the complete working-memory representations and a detailed analysis of the subtasks after the knowledge base stabilized.

The DAA starts with a data flow representation extracted from the algorithmic description. This representation resembles the internal description used by most optimizing compilers, but computer programs manipulate it more easily, and it is felt to be less sensitive when the same algorithm appears in a variety of writing styles. The DAA produces a technology-independent hardware network description. This description is composed of modules, ports, links, and a symbolic microcode. The modules can be registers, operators, memories, and buses or multiplexers with input, output, and bidirectional ports. The ports are connected by links and are controlled by the symbolic microcode.

The DAA uses a set of temporally ordered subtasks to perform the synthesis task. It begins by allocating the base variable storage elements -- constants, architectural registers, and memories with their input, output and address registers -- to hardware modules and ports. Then a data-flow BEGIN/END block, or VT-body,[†] is picked, and the synthesis operation assigns minimum delay information to develop a parallel design. Next, it maps all data-flow operator outputs not bound to base-variable storage elements to register modules. Last, it maps each data-flow operator, with its inputs and outputs to modules, ports, and links. In doing so, the DAA avoids multiple assignments of hardware links; it supplies multiplexers where necessary. These last two mapping steps place the algorithmic description in a uniform notation for the expert analysis phase that follows.

[†] VT-bodies are fully described in Chapter 4.

The expert-analysis subtask first removes registers from those data-flow outputs where the sources of the data-flow operator are stable. Operators are combined to create ALUs, according to cost and partitioning information across the allocated design. The DAA also examines the possibility of sharing non-architectural registers. Where possible, it performs increment, decrement and shift operations in existing registers. Where appropriate, it places registers, memories, and ALUs on buses. Throughout this subtask, constraint violations require trade-offs between the number of modules and the partitioning of control steps. The process is repeated for the next data-flow BEGIN/END block.

The DAA is implemented as a production system via the OPS5 KBES writing system. The KBES tool is based on the premise that humans solve problems by recognizing familiar patterns and by applying their knowledge in the current situation. The tool formulated a problem by using three major components: a working memory, a rule memory, and a rule interpreter. Chapter 2 describes KBESs and provides additional examples.

3.3.1 The working memory. The working memory is a collection of elements that describe the current situation. The elements resemble the records in conventional programming languages:

```
literalize module
  id: adder.0
  type: operator
  atype: two's complement
  bit-left: 17
  bit-right: 0
  attribute: +
```

This working-memory element describes an operator module *adder.0*, which can perform two's complement addition on 18 bits of binary data.

3.3.2 The rule memory. The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:

IF:

the most current active context is to create a link
and the link should go from a source port to a destination port
and the module of the source port is not a multiplexer
and there is a link from another module to the same destination port
and this other module is not a multiplexer

THEN:

create a multiplexer module
and connect the multiplexer to the destination port
and connect the source port and destination port link to the multiplexer
and move the other link from the destination port to the multiplexer

This rule recognizes situations in which a multiplexer needs to be created to connect one port to another.

Each subtask in the DAA is associated with a set of rules for carrying out the subtask. An example of a rule for the fourth subtask appeared above. Most of the rules, like the example above, define situations in which a partial design should be extended in some particular way. These rules enable the DAA to synthesize an acceptable design by determining, at each step, whether a certain design extension respects constraints.

3.3.3 The rule interpreter. The rule interpreter pattern matches the working-memory elements against the rule memory, to decide what rules apply to the given situation. The rule-selection process is data driven; the rule interpreter looks through the rule memory for a rule whose antecedents match elements in the working memory. This is also called forward chaining or antecedent reasoning. The consequences of the rule are applied, and the process is repeated until no more rules apply or until a rule explicitly stops the process. If more than one rule applies, the rule dealing with the most current working memory is

selected first. If multiple rules are still applicable, the most specific rule is selected. This selection mimics following a train of thought, as far as possible, and uses special-case knowledge before general-purpose knowledge.

The separation of expert knowledge from the reasoning mechanism makes the incremental addition of new rules and the refinement of old ones easy because the rules have minimal interaction with one another. By using a KBES approach, DAA uses a weak method⁷⁷ called *match* in place of extensive backtracking. DAA uses *match* to explore the space of possible designs by extending a partial design from an initial state to a final state without any backtracking. DAA proceeds through its major tasks in the same order for each problem; it never varies the order and it never backs up in any problem. This means that at any intermediate state, DAA can determine how to extend the design to achieve an acceptable result. The advantages of using the KBES approach are the ease of validating the knowledge gathered from interviews with experts, the ease of incrementally adding to the knowledge base, the ability to query that knowledge during the design task, and the replacement of extensive backtracking by domain-specific knowledge techniques. The disadvantage is the difficulty of extracting knowledge from the experts.

3.4 Knowledge Acquisition Interviews

The prototype DAA system had about 70 rules and could design a MOS Technology Incorporated MCS6502 microcomputer in about three hours of VAX 11/750 CPU time. We asked many expert designers at INTEL and AT&T Bell Laboratories to critique the design by explaining what was wrong, why it was wrong, and how to fix it. After each critique, rules were modified, new rules were added, and the MCS6502 was re-designed. Based on the critiques, the development DAA system now has over 300 rules, and has designed a

much better MCS6502 microcomputer in about five hours of VAX 11/750 CPU time. In retrospect, clearly much of what we learned was common-sense design knowledge, the same things human designers learn through apprenticeship. The DAA has undergone many improvements and produced many designs of the MCS6502 microcomputer. Below, we illustrate a few of these changes.

Each knowledge acquisition interview began by giving the designer a drawing of the design with a sheet of clear plastic over it. Before the designer started the critique, pieces of cardboard were placed over the design. As the designer proceeded, a piece of cardboard had to be lifted, the plastic written on, and covered by new plastic to correct the design. This provided a complete record of where the designer was focusing attention and what was corrected. The designers found this elicitation procedure compatible with their normal spatial mode of operation. The first prototype DAA system was used to produce the design summarized in Column 1 of Table 4. Each row shows the bits of the specified operator or register type found in the design.† The expert criticism is summarized in four points:

- Operators of different types and sizes should be combined into ALUs.
- One-bit operators within the same block should not be combined, because multiplexers are more expensive than most one-bit modules.
- Registers should increment, decrement, and shift their values internally, where possible.
- Temporary registers to the controller should be eliminated, and one latched register

† A figure is not provided because it is totally inscrutable.

should be placed in front of the controller.

Table 4. MCS6502 – THREE DESIGNS

Designs	1	2	3
And	20	20	20
Cmp	177	1	1
Minus	64	0	0
Or	9	9	9
Not	21	21	21
Plus	540	0	0
Shifts	35	1	1
Xor	9	9	9
Alu	0	35	35
Dreg	450	281	210
Treg	1227	0	62
Max In	2122	2637	473
Max Out	293	377	84
Bus In	0	0	769
Bus Out	0	0	210

The rules were changed to produce the design summarized in Table 4, Column 2, and illustrated in Figure 1. To produce this design, partitioning information was added, based on connectivity of data paths and similarity of operators among blocks. The details of the partition and cost estimators is given in Chapter 5. This simplified the decision about which modules to combine when hardware operators are shared among abstract operations detailed in the algorithmic description. Rules were also added to combine modules of different sizes and types. As Column 2 shows, the ALU number increased, decreasing the plus, minus, shift, and compare numbers. Rules were also added to decrease the amount of temporary register storage.

Figure 1 shows the eight-bit data paths of the MCS6502. The one-bit and 16-bit data paths were omitted for clarity. Each of the symbols represents a module. The circles are

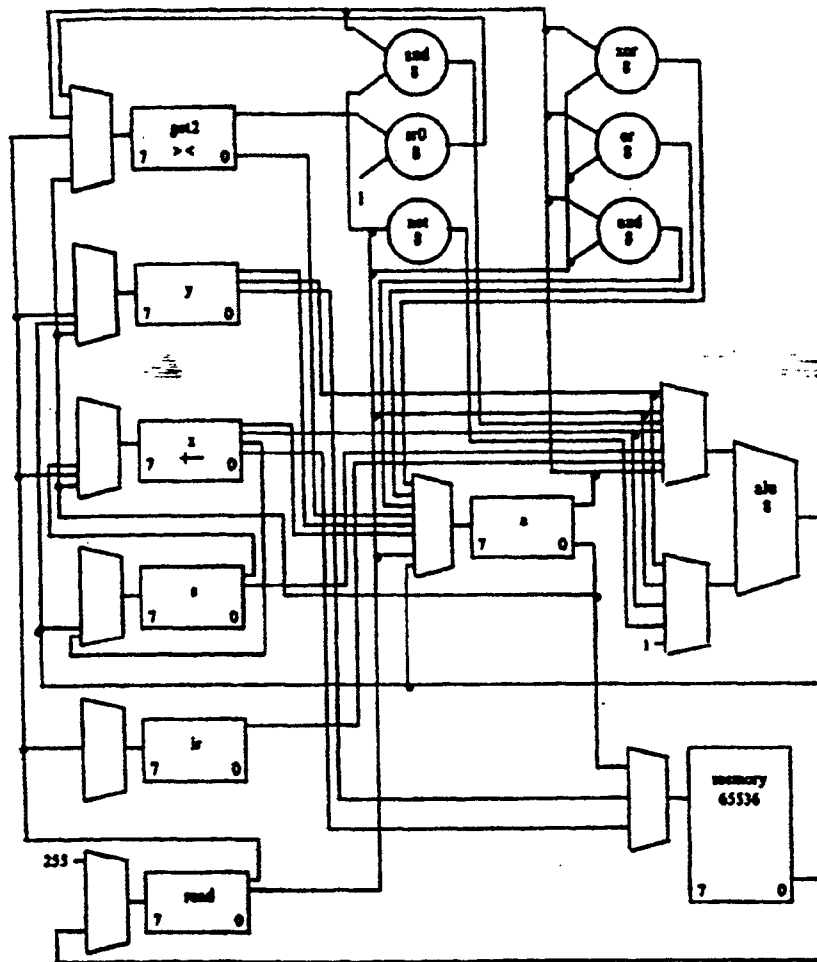


Figure 1. MCS6502 -- 8 BIT MUX DATA PATHS

single function ALU modules to AND, SHIFT, NOT, XOR, and OR data, the small trapezoids are multiplexers that gate one of their inputs to the output, the small rectangles are registers, the large rectangle is the memory, the large trapezoid is a multi-function

33

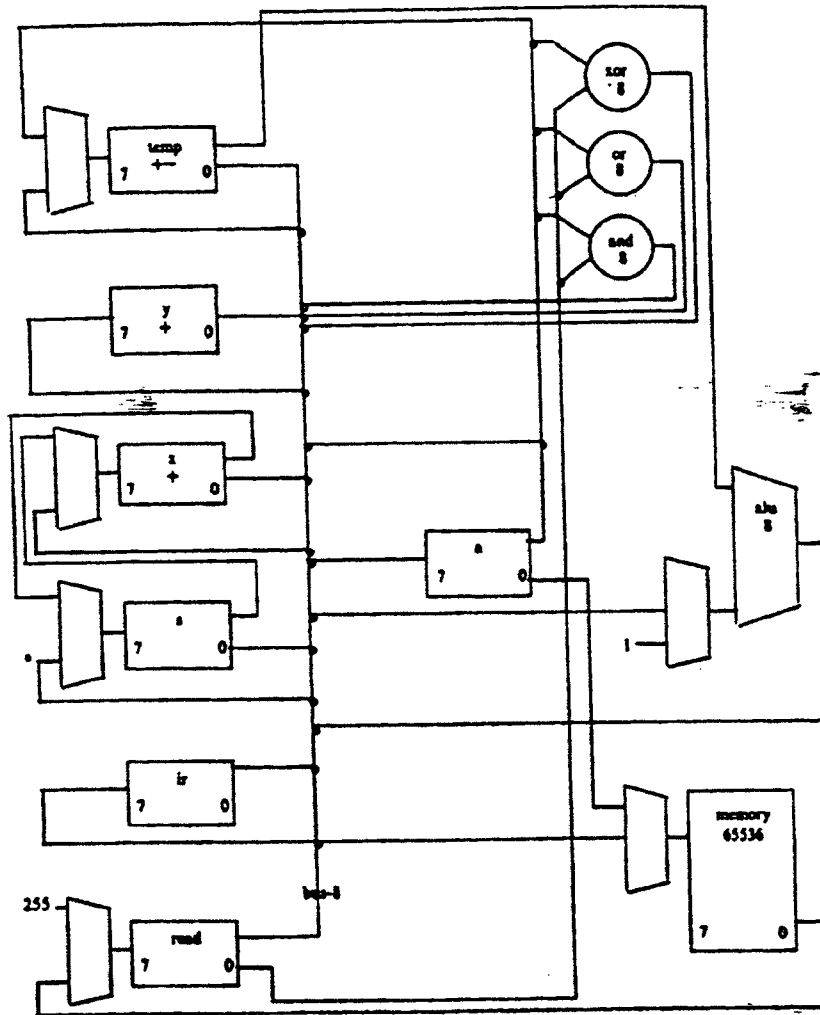


Figure 2. MCS6502 - 8 BIT BUS DATA PATHS

ALU, and each of the lines represents a link between the modules. Where the links join with the modules, a port is defined. An obvious problem, pointed out by our experts, was

overuse of multiplexers. They suggested ways of distributing the multiplexer hardware to form buses.

The rules were changed in response to these critiques, resulting in the design of Table 4, Column 3, and Figure 2. To produce this design, rules were added to recognize when a multiplexer should be converted into a bus and how to share that bus with other distributed multiplexers. In addition, new rules decreased the amount of declared register storage. Specifically, registers were not needed to multiplex information into the data flow BEGIN/END blocks. As Column 3 shows, the multiplexer numbers decreased, increasing the bus numbers. The declared register number also dropped.

Though this design was acceptable to our experts, it was not perfect. Further changes led to improvements such as multiple buses of different widths. However, these changes did not affect the MCS6502 because it did not require multiple buses. This brings up an interesting point about expert systems: they are never totally finished. Like human designers, the DAA becomes a better designer as its rule memory expands. Until all possible world knowledge about designing microprocessors has been codified in the DAA's rules, there will always be room for improvement in its designs. However, for the data representation, knowledge representation, and System/370 experiment discussions in Chapters 4, 5, and 6, the rules were held constant.

3.5 Analysis of Knowledge

The last step in extracting knowledge is to examine it and determine what has been learned. One merit of codifying knowledge in a KBES is that it can be easily quantified and qualified. Chapters 4 and 5 provide analysis by examining the representations and functional knowledge in the DAA. This section provides additional analysis by examining

both the extent to which each rule embodies domain knowledge and the type of each rule. The analysis includes the selection criteria of types, the English translation of several rules, and a brief discussion of the translated rules. These translations must be taken with a grain of salt because rules do not stand by themselves, but are part of a larger network of rules connected by the working memory and the inference engine.

Table 5. RULES BY KNOWLEDGE TYPE

Domain Specific			Domain Independent		
Type	Rules	Firing	Type	Rules	Firing
Design	151	3768	Cleanup	33	1586
Context	18	1562	Input	9	64
Setup	70	70	List	14	1290
			Test	19	203
Total	239	5400	Total	75	3143
Total DAA	314	8543			

The rules in DAA either contain domain specific or domain independent knowledge. Knowledge directly related to the implementation design task is domain specific, while knowledge of a more general type is domain independent. An example of domain-specific knowledge is how to make particular design decisions and an example of domain-independent knowledge is how to count things. Distributed between these two categories DAA contains seven distinct rule types. Rules that extend the partial design, control the context or focus of attention, and maintain the technology-independent database are domain specific. Rules that remove unneeded working-memory elements, transform the input description, simulate list and set operations, and simulate procedure calls or calculations in antecedents are domain independent.

Table 5 summarizes these categories by listing the number of rules in each category

and the number of rule firings for the complete design of the SCF3 processor. The structured control flow processor,⁷⁸ SCF3, was chosen partly because it is slightly smaller than the MCS4502 design (two hours of VAX 11/780 CPU time), but mostly because it is being carried through the CMU/DA system to produce a working silicon chip. Table 5 shows that three quarters of the rules and two thirds of the rule firings are domain specific. Most of the domain specific rules and firings are used to extend the partial design (See Row Design in Table 5). The remaining rules and rule firings are domain independent or overhead rules. Most of these rules and firings are used to clean up unneeded working-memory elements and simulate list and set operations (See Rows Cleanup and List in Table 5).

3.5.1 Rules that extend the design. This category of rules can be distinguished from other rules in DAA by their modification action to the design. They either make design decision directly, or use a test rule to make design decision for them.

IF:

the most current active context is bus allocation
and there is a link from a module to a bus
and there is a link from the bus to a multiplexer
and there is a link from the module to the multiplexer

THEN:

check to see if the bus is idle during the control step needed
to multiplex the values

Figure 3. MODULE-TO-MUX-AND-BUS-WITH-BUS-TO-MUX

The rule in Figure 3 extends the partial design by recognizing the need to use a bus to make a connection and creates a context for a test rule to see if the bus is idle. If the bus is idle the connections are moved from the multiplexer to the bus. This rule uses a test rule to complete a match. If OPS5 could call procedures during the match cycle, this rule and the appropriate test rule would have been merged. The rule in Figure 4 extends the partial

IF:
 the most current active context is to allocate value trace operators
 and there is a value trace + operator
 and one input to the + operator is a register
 and the other input is the constant one
 and the output of the + operator is the input of a bit read operator
 and the output of the bit read is the full width of the register feeding
 the + operator

THEN:
 add the increment attribute to the register

Figure 4. ASSIGN-INC

design by recognizing the need for an incrementer and assigns the increment function to the correct register. This is an example of a rule that directly extends the design.

IF:
 the most current active context is fold allocation
 and there is an unbound module whose width is greater than 1
 and there is a database operator for this module whose group is either
 ARITHMETIC, RELATIONAL, SHIFT, or ALU
 and there is a second database operator whose group is either
 ARITHMETIC, RELATIONAL, SHIFT, or ALU
 and there is a bound module whose operator is in the set of the second
 database operator with the same width as the first module

THEN:
 check to see if the proximity and cost factors show they should be
 combined

Figure 5. BIND-ARITHMETIC-OPERATORS-SAME-SIZE

The rule in Figure 5 extends the partial design by recognizing when to try to combine an unbound module and a bound hardware module by creating a context for a test rule. If the proximity or cost value exceeds the minimum required value, then the test rule combines the modules. This rule shows an important use of the technology database as part of the rule. It could have been written as a number of separate rules using constants to link the type of bound and unbound modules together. This would have increased the number of rules, decreased the CPU time to run the system, and increased the difficulty of modifying the rule base. This general style of writing rules is less efficient in computer time, but more efficient

in rule-base modification time.

3.5.2 Rules that control the context or focus attention. This category of rules can be separated from other rules in DAA because they only manipulate the context status. By manipulating active, pending, next and not-satisfied status codes, these rules control the focus of the design task. They also provide the ability to sequence a list of tasks.

IF:
 there is a request to allocate a value trace body
THEN:
 remove the request
 and make the most current active context to allocate temporary variables
 and make the next pending context to allocate control steps
 and make the next pending context to allocate operators
 and make the next pending context to fold allocation

Figure 6. ALLOCATION

The rule in Figure 6 generates a number of pending contexts and thus controls the focus of attention of DAA. The pending contexts are modified into active contexts when there are no other rules to fire in the currently active context. This helps minimize the set of rules that could fire and decreases the number of internal match states in OPS5. This is an example of one of six rules that set up a list of subtasks.

IF:
 the most current context is a next context
THEN:
 make it the most current active context

Figure 7. NEXT

The rule in Figure 7 modifies a waiting context to an active state and thus controls the focus of attention of DAA. About ten percent of the total rule firings are composed of this rule. At first glance it seems to be an unnecessary rule, and indeed it is! However, using it reduces the number of rules that can fire, which decreases the total CPU time in OPS5. The

design time for the MCS6502 was reduced by four percent. This rule should be removed and the other rules modified if parallelism is measured on DAA. This is an example of one of three rules that monitor the active context and change context if appropriate, thus limiting the set of active rules.

3.5.3 Rules that manage the technology-independent database. The rules in this category are distinguished from other rules in DAA by their default database creation actions.

IF:
 the most current active context is to declare variable allocation
 and ~~there~~ is no database operator PLUS
THEN:
 make a database operator with hardware opcode PLUS,
 control step delay 1, value trace opcode +, 0 modules
 currently allocated, a maximum of 999 modules, and of group
 ARITHMETIC

Figure 8. DEFAULT-PLUS

The rules in Figures 8 and 9 set up the technology-independent database. They fire once if the user has not provided the appropriate constraint value. Constraint information is kept in working memory and used by the rules as bound variables.

IF:
 the most current active context is to declare variable allocation
 and there is no database fold parameter for ARITHMETIC types
THEN:
 make a database fold parameter with type ARITHMETIC,
 proximity and cost values of 0.500

Figure 9. DEFAULT-FOLD-ARITHMETIC

This technique allows easy change of target technologies and individual fine tuning of design constraints. These are examples of rules that propagate default constraints through working memory to all the design rules.

3.3.4 Rule that remove unneeded working-memory elements. This category of rules can be separated from other rules in DAA by observing that their actions are only to clean up the working memory.

IF:
 the most current active context is to clean working memory associated with
 value trace body *N*
 and there is an outnode associated with value trace body *N*
THEN:
 then remove it

Figure 10. CLEAN-OUTNODES

The rule in Figure 10 removes unnecessary working-memory elements, thus getting rid of unneeded details of the design. This domain-independent rule along with seven others reduces the size of working memory to the current VT-body and the hardware that has been allocated. Without these rules only small designs can run through DAA. This is an example of a domain-independent rule that aids DAA in execution, but does not perform any design task.

IF:
 the most current active context is to declare variable allocation
 and no other rule can fire
THEN:
 then remove the active context

Figure 11. DECLARED-VARIABLE-ALLOCATION-DONE

The rule in Figure 11 removes a currently active context that has been completely satisfied. This rule and several like it do not contribute to the design domain, but are simply aids in debugging the KBES. They aid debugging because there is a demon rule that dumps working memory to a file and lets the user interact with the KBES if an active context cannot be satisfied by any rule. This is an example of a domain-independent rule that helps knowledge engineers debug DAA.

3.5.5 Rules that transform the input description. This category of rules is similar to the rules that extend the design except they manipulate the input description. They either directly modify the input description, or use a test rule to make the change for them.

IF:
 the most current active context is to allocate temporary variables
 and there is a value trace operator of type +, -, or *
 and its output is bit read
 and the offset to the bit read is zero
 and the bit read produces fewer bits than does the value trace operator
THEN:
 make a trim record of the number of bits from the bit read

Figure 12. TRIM-ADD-SUB-MUL-OUTPUT-ZERO-NIL-TRIM

The rule in Figure 12 transforms the input VT description to a better description by minimizing the size of addition, subtraction and multiplication operators. This rule and a rule similar to it will repeatedly fire until the rule in Figure 13 finally limits the operator output size to the largest size required as input by any other operator. This is an example of a domain-independent rule that could have been added to the VT transformation package.

IF:
 there is a trim record for an outnode
 and the outnode has a larger number of bits than does the trim record
THEN:
 set the number of bits for the outnode equal to the trim record

Figure 13. TRIM-OUTPUT

The rule in Figure 13 waits until there are no more modifications of the trim record created by the rule in Figure 12 and then modifies the VT operator. This is an example of a rule associated not with the task of extending the design, but with modifying the input specification.

3.5.6 Rules that simulate set and list operations. This category of rules can be separated from other rules in DAA because it modifies the list working-memory element.

IF:
 the most current active context is to step through a list
 and create a new active context with the first object from the list
 and there is an object on the list
THEN:
 shift the first element off the list
 and make a new active context with the first element of the list
 as its object

Figure 14. NEXT-LIST-1

The rule in Figure 14 steps through a list of working-memory elements and creates a new goal for every element in the list. For example, this rule sequentially assigns control-step information to a list of VT operators.

IF:
 the most current active context is to step through a list
 and create a new active context with the first object from the list
 and there not is an object on the list
- THEN:
 remove the active goal of next list

Figure 15. END-LIST-1

The rule in Figure 14 will stay active until the rule in Figure 15 recognizes there is nothing more to do and removes it. The usage of these rules increases for bigger designs because larger designs have longer lists to manage. These rules are examples of domain-independent rules that simulate set operations on a list of working-memory elements.

3.5.7 Rules that simulate procedure calls and calculations for antecedents. This category of rules can be distinguished from other rules in DAA because they only test the results of a procedure call or computation of a previous rule. The rule in Figure 16 combines two registers if the result of a layout estimation shows that they would be placed within a

IF:
 the most current active context is to fold two registers together with
 a given cost and proximity value
 and the proximity value exceeds the minimum required proximity value
THEN:
 combine the two registers

Figure 16. FOLD-REGISTER-PROX

database minimum value of one another. If the result of a stability calculation shows that a register is required between control steps, the rule in Figure 17 marks it not stable.

IF:
 the most current active context is check the stability of a register
 and the register is not stable
THEN:
 then mark the register not stable

Figure 17. NOT-STABLE-REGISTERS

These are examples of domain-independent test rules because they extend the ability of OPS5 to do procedure calls or calculations in the antecedent part of the rule. The domain-dependent part of the knowledge is provided by the rule that activated them.

3.6 Development Summary

Chapter 3 has overviewed the development cycle of the DAA system. It has discussed the case-study interviews, which led to the prototype system, and the acquisition interviews that have further developed the system. Using a KBES approach has allowed the incremental addition of modular knowledge and queries about that knowledge during the design task.

During the case-study interviews the designers discussed the global picture, partitioning, selection, and allocation tasks. They began with a high-level overview of the hardware, which listed inputs and outputs to the outside world, the functions the hardware should provide, general constraints, and design feasibility with consideration of the target

technology. They generally partitioned the global picture into smaller blocks and emphasized minimizing connections among blocks, selecting blocks that operated as parallel or serial units, and grouping according to similarity of function. Partitions were chosen for allocation in a decreasing order of difficulty or degree of constraint. The designers reasoned that if the most difficult part could be designed, the rest of the design was feasible.

Once a partition was selected for allocation, it was carried out either in parallel or in series. A parallel design made thinking of the control logic much simpler, while a serial design minimized the design area. The constraints of the parallel design were examined for size violations to determine the parts to be serialized by adding data paths, registers, and control logic to the initial parallel design. The constraints of the serial design were examined for speed violations to determine the parts to be reimplemented in parallel. If the designers recognized a part of the design as similar to a part of a previous design, they used what they knew had worked in the past. Within each partition, designers allocated clock phases, operators, registers, data paths, and control logic. The order was interesting because once registers and data paths were allocated, they were not changed. The control was changed because it was the hardest thing to think about and because it depended on a constant structure for the data-path elements.

The designers described the iteration process as a step-by-step refinement to meet violated constraints. They looked for a technology change to meet a constraint before making a design change. This could be as simple as finding a new chip in the TTL data book or as complicated as a design-rule shrink. Next, they would sacrifice functionality to meet a constraint. The relative importance of constraints is application dependent. The designers mentioned the constraints of speed, area, power, schedule, cost, drive capabilities, and bit width. Other design changes consisted of global improvements not recognized until

the design neared completion. This suggests that the general choice of partitions and the initial design style selections approached optimum and that designers do not seem to use much backtracking in their designs.

A prototype system was built using the knowledge from the interviews and available book knowledge. The prototype system has been transformed into a development system through hundreds of hours of knowledge acquisition interviews with expert VLSI designers. In retrospect, much of what we learned was common-sense design knowledge, the same things human designers learn through apprenticeship. This learning process is not complete, nor will it ever be complete. Like human designers, the DAA becomes a better designer as its rule memory expands. Until all possible world knowledge about designing microprocessors has been codified in the DAA's rules, there will always be room for improvement in its designs.

Chapter 4: DAA REPRESENTATIONS

The DAA synthesizes a technology-independent representation of memories, registers, operators, data-paths and timing signals from an algorithmic description of a VLSI system. The two major components of any program are data and control structures. The DAA decomposes the problem into three data-structure representations and four control tasks. The representations are defined by the algorithmic-description language, VT, the constraints placed on the design, and the technology-independent-hardware-network description language, SCS. The tasks involve allocating global-storage elements, timing information, local-storage and processing elements, applying global improvements, and routing connections. They transform the VT representation and constraints into SCS using partition and cost estimators to provide abstract high-level floor-planning information. This problem division has been identified by designers and used to limit the complexity of the task of choosing implementations as discussed in Chapter 3. This chapter discusses the three representations and the bookkeeping information used by the DAA. Chapter 5 outlines the knowledge used by the DAA to design VLSI systems.

4.1 Algorithmic Representation

In the CMU/DA system, the input description is written in ISPS,¹⁶ a hardware-description language that is similar in many ways to programming languages such as ALGOL or Pascal. An ISPS description consists of a series of declarations of *entities*. Some of these are simple *carriers*, which hold the data being manipulated, similar to variables in a programming language. Other entities are procedures or functions much like the procedures or functions of a programming language. These define how the data is manipulated. The procedures are specified by data *operators* that do arithmetic, logical,

relational, and shift operations on the data, and by sequential, parallel, and conditional constructs that show how the data operators are combined.

```

Example :=
Begin

  ** Storage.Declaration **

  cpage\current.page<0:4>,
  i\instruction<0:11>,
    pb\page.0.bit<>      := i<4>,
    pa\page.address<0:6> := i<5:11>

  ** Address.Calculation **

  Global cadd\effective.address<0:11> :=
  Begin
    Decode pb =>
    Begin
      0 := cadd - '00000 @ pa,
      1 := cadd - cpage @ pa
    End
  End

End

```

Figure 18. SAMPLE ISPS DESCRIPTION

The ISPS description is considered an algorithmic description, not a representation of the structure of the implementation. Carriers do not necessarily represent individual registers. A carrier that simply holds temporary values, for example, might be implemented by a bus, which simply transfers values to the next stage without storing them, or several carriers may share the same physical register. On the other hand, several instances of the same logical entity, such as an accumulator or program counter, might be required at the same time in a highly parallel implementation such as a pipelined machine. Here it would take several registers to implement the same carrier.

A simple ISPS example description is shown in Figure 18. This ISPS fragment is from the description of the Digital Equipment Corporation PDP-8. It first defines the

48

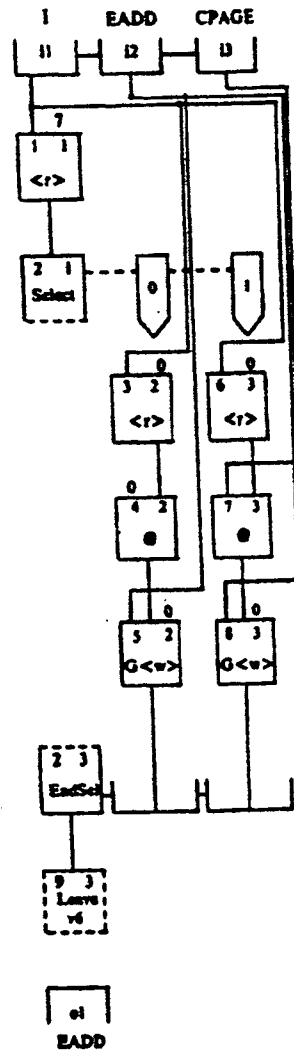


Figure 19. SAMPLE VT

current page and the instruction carriers. It then labels specific fields of the instruction carrier as the page-zero bit and page-address carriers. The last part of the ISPS fragment decodes the page-zero bit of the instruction carrier and sets the effective-address carrier equal to the page-address carrier concatenated with either zero or the current-page carrier.

The ISPS description is compiled into a VT^{17,18} data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called *operators* and correspond to operations that take certain values as input and produce new values as output. They are translations of the ISPS unary and binary operations, operations that change or access fields in words or words in arrays, control operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called *outnodes* and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called *VT-bodies*, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops.

The VT fragment in Figure 19 shows the same decoding loop as the ISPS description in Figure 18. Each of the blocks represents a VT operator. For example, the operator \oplus , called a *concatenate*, produces the concatenation of its input values. The operator $\langle r \rangle$, called a *bit read*, produces the subfield of its left input that begins with the bit specified by its right input, and continues to the left for the width specified by the output. The operator $G\langle w \rangle$, called a *global bit write*, produces the left input, with the middle input overwriting it, starting at the bit position specified by the right input. The compound operator

50

Table 6. VTBODY AND OUTNODE

Attribute	Values
Literalize vtbody	
id	id number
type	s=slist, r=carrier, m=mapping, v=vtbody
parent	id of vtbody declared in
entry-flag	vector: can be process, global, main, map, vtbody, slist-head, bit, word, external, repeat, multiplied, divided and loop
word-left	starting word
word-right	ending word
bit-left	starting bit
bit-right	ending bit
isp-name	from isps compiler
inputs	name of input outnode list
operators	name of operator list
outputs	name of output outnode list
calls	name of vtbody, or slist we call list
opcalls	name of operator list that call, enter, leave, restart, or resume this vtbody
map-id	mapping onto id
map-word-scale	mapping scaling factor
map-word-offset	mapping word offset from lowest numbered word in target
map-bit-offset	mapping bit offset from right most bit of the target mapped word
attributes	name of attribute list
qualifier-trees	name of qualifier tree
control-steps-assigned	control steps have been assigned for this vtbody
vt	vtbody number
Literalize outnode	
id	id number
type	f=formal parameter, i=vtbody input, c=constant o=vtbody output, p=value from operator node
carrier-id	initially from isp
bits	number of bits
isp-name	name from isps compiler
value	value if a constant
vt	vtbody number

consisting of the blocks *SELECT* and *ENDSEL*, and the items connected to them by dashed lines, is called a *SELECT*. It is used in the VT to implement the DECODE and IF

51

Table 7. OPERATOR, BRANCHES, LISTS AND TREES

Attribute	Values
Literalize operator	
id	id number
opcode	operator
type	TC, OC, SM, US (from ISPS)
call	name of vibody or alist we call, pstart restart, resume, or leave
inputs	name of outnode input list
outputs	name of outnode output list
attributes	name of attribute list
qualifier-trees	name of qualifier tree
parent	name of parent vibody
branches	name of branch list for select and diverge
control-step-begin	starting control step number
control-step-end	ending control step number
vt	vibody number
Literalize branch	
id	id number
type	b=select, d=diverge
parent	parent operator
activations	name of list of activations for selects
operators	name of list of operators
inputs	name of list of inputs for selects
qualifier-trees	name of qualifier tree
vt	vibody number
Literalize lists	
id	id number
list	vector: list elements
vt	vibody number
Literalize trees	
id	id number
list	vector: list elements
tree	name of next tree
vt	vibody number

... THEN ISPS constructs. In this example, it implements the decoding of the instruction register from Figure 18. Alternate actions are shown side-by-side with the value of the selector required for each shown at the top. The result of the chosen action is then passed

to the ENDSEL where it continues to the LEAVE operator. The LEAVE operator exits the VT-body and passes its input back to the calling VT-body.

For each design task the DAA is given a working-memory representation of the VT, shown in Tables 6 and 7. The first column provides the name of the attribute, while the second column describes the possible values. An attribute that requires a variable-length list of values is either represented as a vector, denoted by *vector*, or as the name of the list working-memory element that contains the values. This inconsistency is brought about because OPS5 can represent only one vector quantity in each working-memory element. Furthermore, the non-hierarchical limitation of the OPS5 working-memory representation increases the time required to match the rules because the graph is represented as pointer attributes to working-memory elements. The working-memory list of the ISPS and VT decoding loop in Figures 18 and 19 is given in Appendix A.

4.2 Technology Database and Constraints

The second input representation provides technology-sensitive information and design constraints. The information is referred to as technology sensitive rather than technology independent because it is sensitive to a particular technology rather than independent of technology considerations. The technology database provides the name translation between VT and SCS operators, and the number of micro-control steps required for the operation.[†] It also provides constraints on the number of each module the DAA may use in the design, thus providing rough limits on area and power. No constraints are given for speed because

[†] The notion of micro-control steps is further explained in the Section 3.3.2, which describes operator assignment to control steps.

Table 2. USER PARAMETERS

Attribute	Values
literalize db-operator	
hw-opcode	hardware operator code
control-step-delay	amount of delay
vt-opcode	value trace operator code
allocated	how many have I allocated so far
maximum	how many can I allocate
group	what group do the operators belong to
literalize max-delay-per-control-step	
delay	number of delay units per control step
literalize fold	
type	ARITHMETIC, LOGICAL, REGISTER
prox	proximity
cost	cost

the DAA starts with the maximally parallel design, only serializing it as it hits area and power constraints. The technology database provides information about how many micro-control steps comprise a full-control step. This information is given in the working-memory element *max-delay-per-control-step*. The last technology-sensitive inputs are the proximity and cost thresholds for arithmetic, logical and register modules. They regulate how close a fit is required between two modules before they will be combined. Their use is further explained in Sections 5.4.2 and 5.4.4, which describe register and module allocation. Table 8 shows the working-memory representation for these parameters. If the user does not supply values for these parameters in a file named *daa.l*, the values from Tables 9, 10, and 11 are used. Delay values of -1 are a special case of clock synchronization discussed in Section 5.3.2. Because these values were suggested by designers of MOS microprocessors attempting to design fast processors, the use of hardware is unlimited. The working-memory list of this table is given in Appendix B.

Table 9. DEFAULT DB-OPERATOR -- PART I

Hardware Opcode	Delay	VT Opcode	Maximum	Group
PLUS	1	+	999	ARITHMETIC
MINUS	1	-	999	ARITHMETIC
MULT	1	*	999	ARITHMETIC
DIV	1	/	999	ARITHMETIC
MOD	1	MOD	999	ARITHMETIC
UMINUS	1	--	999	ARITHMETIC
AND	1	AND	999	LOGICAL
EQV	1	EQV	999	LOGICAL
OR	1	OR	999	LOGICAL
XOR	1	XOR	999	LOGICAL
NOT	1	NOT	999	LOGICAL
EQL	1	EQL	999	RELATIONAL
NEQ	1	NEQ	999	RELATIONAL
LSS	1	LSS	999	RELATIONAL
LEQ	1	LEQ	999	RELATIONAL
GEQ	1	GEQ	999	RELATIONAL
GTR	1	GTR	999	RELATIONAL
TST	1	TST	999	RELATIONAL
SRO	1	SRO	999	SHIFT
SRI	1	SRI	999	SHIFT
SRD	1	SRD	999	SHIFT
SRR	1	SRR	999	SHIFT
SLO	1	SLO	999	SHIFT
SLI	1	SLI	999	SHIFT
SLD	1	SLD	999	SHIFT
SLR	1	SLR	999	SHIFT
SLI	1	SLI	999	SHIFT
SRI	1	SRI	999	SHIFT
DELAY	0	DELAY	999	SYNCHRONIZATION
WAIT	0	WAIT	999	SYNCHRONIZATION
DWAIT	0	TIME WAIT	999	SYNCHRONIZATION
STOP	0	STOP	999	SYNCHRONIZATION

Table 10. DEFAULT DB-OPERATOR - PART 2

Hardware Opcode	Delay	VT Opcode	Maximum	Group
SPAD	0	PADO	999	WIRING
SPAD	0	PADS	999	WIRING
FREAD	0	BIT-R	999	WIRING
FWRITE	0	BIT-W	999	WIRING
AREAD	-1	WORD-R	999	WIRING
AWRITE	-1	WORD-W	999	WIRING
CONCAT	0	@	999	WIRING
GFREAD	0	GBIT-R	999	WIRING
GFWRITE	0	GBIT-W	999	WIRING
GAREAD	-1	GWORD-R	999	WIRING
GAWRITE	-1	GWORD-W	999	WIRING
SELECT	-1	SELECT	999	BRANCH
ENDSEL	0	ENDSEL	999	BRANCH
DIVERGE	0	DIVERGE	999	BRANCH
MERGE	0	MERGE	999	BRANCH
ENTER	-1	ENTER	999	CONTROL
CALL	-1	CALL	999	CONTROL
PSTART	-1	PSTART	999	CONTROL
LEAVE	-1	LEAVE	999	CONTROL
RESTART	-1	RESTART	999	CONTROL
RESUME	-1	RESUME	999	CONTROL
TERMIN	-1	TERMIN	999	CONTROL
NOOP	0	NO.OP	999	NOOP
UNDEF	0	UNDEF	999	UNDEFINED
UNPREDICTABLE	0	UNPREDICTABLE	999	UNPREDICTABLE
PARITY	1	PARITY	999	PARITY
IS.RUNNING	0	IS.RUNNING	999	IS.RUNNING
INC	1	INC	999	INC
DEC	1	DEC	999	DEC
CLEAR	1	CLEAR	999	CLEAR
MUX	0	MUX	999	MUX
DEMUX	0	DEMUX	999	DEMUX
BUS	0	BUS	999	BUS
ALU	1	ALU	999	ALU

Table 11. DEFAULT FOLD AND DELAY

Fold Type	Proximity	Cost
ARITHMETIC	0.5	0.5
LOGICAL	0.5	0.75
REGISTER	0.5	0.5
Max Delay Per Control Step		50

4.3 Technology-Independent Hardware Network

The VT representation and the constraints are used by the rules described in Chapter 5 to create a technology-independent, but technology-sensitive implementation description. In the CMU/DA system, the technology-independent description is given in a structure and control specification language, SCS¹⁹ (soon to be replaced by DIF).²⁰

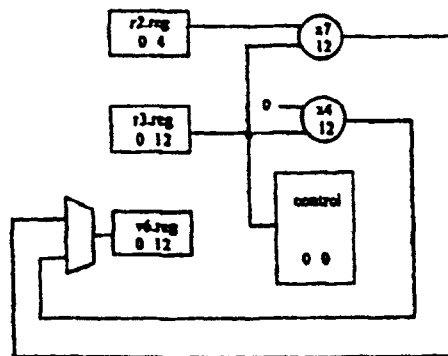


Figure 28. SAMPLE STRUCTURAL SPECIFICATION

The structural specification models the functional and logical structural levels as a network of modules. The modules communicate through links connected to module-interface ports. The ports may sink input, source output, or both sink input and source output. The control specification describes the control sequences that govern the behavior of the network.

57

```

controller: controller0;
EADD {v6} :
  cstep [1:1] {
    select controller0 (controller0.source0) {
      case [0] {
        cstep [2:2] {
          mux v6.input.mux (v6.input.source1)
            {v1_references: v6.x5};
          gfwrite v6.x4 {v1_references: v6.x5};
          concat v6.x4 {v1_references: v6.x4};
        }
      }
      case [1] {
        cstep [3:3] {
          mux v6.input.mux (v6.input.source0)
            {v1_references: v6.x8};
          fwrite v6.register {v1_references: v6.x8};
          gfwrite v6.x7 {v1_references: v6.x8};
          concat v6.x7 {v1_references: v6.x7};
        }
      }
    } {v1_references: v6.x2};
  }
  cstep [3:3] {
    leave EADD {v1_references: v6.x9};
  }
end;

```

Figure 21. SAMPLE CONTROL SPECIFICATION

Figure 20 shows a graphical SCS for the decoding loop of Figures 18 and 19. Each of the symbols represents a module. The circles are single function wiring modules to bring together or concatenate two sets of signals, the trapezoid is a multiplexer that gates one of its two inputs to its output, the small rectangles are registers, the large rectangle is the controller, and each line represents a link between the modules. Where the links join with the modules, a port is defined. Because the initial ISPS was so simple, the final design is just some wiring, the specified architectural registers, a multiplexer and the controller.

Table 12. HARDWARE

Attribute	Values
Literalize module	
id	id number
type	REGISTER, MEMORY, CONSTANT, OPERATOR
stype	TC, OC, SM, US
word-left	starting word
word-right	ending word
bit-left	starting bit
bit-right	ending bit
value	value for constants
opt-flag	optimization flag
Literalize port	
id	id number
number	source number 0 or 1 nil otherwise
type	INPUT, OUTPUT, BIDIRECTIONAL
bit-left	starting bit
bit-right	ending bit
module	id of module
Literalize link	
source-port	id of source
source-bit-left	bit number of source starting bit
source-bit-right	bit number of source ending bit
dest-port	id of destination
dest-bit-left	bit number of destination starting bit
dest-bit-right	bit number of destination ending bit

Figure 21 shows the control specification for the structural specification in Figure 20. Each control step, *cstep*, has a list of ports to activate with references to the VT operators that cause the activations. Branches in the control specification are shown by *select* operations on controller inputs with *case* statements to show possible values and actions. Thus, this specification shows that if the controller0.source0 input is 0, then the bottom input to the multiplexer is turned on, writing into the global variable v6, using the output of the concatenate, v6.x4. If the controller0.source0 input is 1, then the top input to the multiplexer is turned on, writing into the global variable v6, using the output of the

concatenate, v6.x7.

For each design task the DAA generates a working-memory representation of the SCS representation, shown in Table 12. The first column provides the name of the attribute, while the second column describes the possible values. The way this representation is created and manipulated is discussed in Chapter 5. The working-memory listing for the SCS representation is given in Appendix C.

4.4 Bookkeeping Information

There are several other miscellaneous bookkeeping working-memory elements used to guide the context of the problem solving, to count the allocated multiplexer and bus ports, to remember the state of control-step assignment, and to remember the minimum size for VT operators.

The most important is the *context* working-memory element, which directs the focus of the DAA to a particular task. For example:

```
(context 'status active 'operation make-link 'object v6.x4.output 11 0 v6.x4.pl
v6.input.source1 11 0 v6.x5.pl)
```

This is a request for all the rules that make links from one module to another to make the best connection between bits 11 and zero of ports v6.x4.output and v6.input.source1.

The multiplexer, bus, and controller modules require an additional working-memory element to keep track of the allocated number of input and output ports. These working-memory elements should be subsumed into the module working-memory element in future versions of the DAA. The control-step, micro-step, and trim working-memory elements are temporary places to store information during the assignment of control steps to operators and the trimming of excessive bits from add, sub, and multiply operators. The meaning of

Table 13. MISCELLANEOUS

Attribute	Values
Literalize content	
status	pending, active, next, or not-processed
operation	operation goal
object	vector: objects to operate on
Literalize mux-port-count	
module	module of mux
icnt	number of input ports
Literalize bus-port-count	
module	module of bus
icnt	number of input ports
ocnt	number of output ports
Literalize controller-port-count	
module	module of controller
icnt	number of input ports
Literalize control-step	
cs	current control step
Literalize micro-step	
ms	current micro-control step
Literalize trim	
id	outside id to trim
bits	possible bits to trim to

these working-memory elements is fully described in Sections 5.3.2 and 5.3.3, which overview the control assignment and operator trimming. Table 13 shows the working-memory representation of these structures.

4.5 Representation Summary

In summary, Chapter 4 has presented three representations that have spanned the algorithmic to technology-independent-hardware-network levels in the CMU/DA system. The representations are defined by the algorithmic-description language, VT, the constraints

placed on the design, and the technology-independent-hardware-network description language, SCS. Both the modularity and the granularity of the data representations have affected how rules are created for the system. Chapter 5 shows how these representations are used by the DAA to design VLSI systems. It also groups the knowledge together into sections to highlight the modularity of the knowledge in the DAA.

Chapter 5: DAA KNOWLEDGE

Chapter 4 has discussed the VT, constraint, and SCS representations used by the DAA. These representations combined with the knowledge extracted from expert VLSI designers, using the interview method described in Chapter 3, have been used to design many computers including the MCS6502 and the IBM System/370.

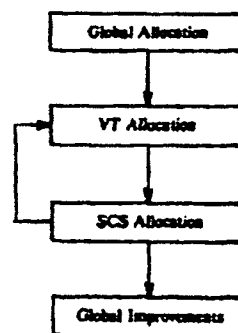


Figure 22. DAA SUBTASKS

An important goal of the research has been to understand how VLSI designers choose computer implementations. The problem division they use, which is shown in Figure 22, is grouped into general service functions, global implementation allocation, VT allocation, SCS allocation, and global improvements to the implementation. These subtasks are implemented in the DAA and discussed by functional sections in this chapter. A summary of those subtasks, rule types, and rule usage for the SCF3 design is provided in Table 14. The chapter also discusses two sets of rules that have been rewritten as procedures for efficiency. They estimate partitioning and cost information used to provide high-level floor plan information to the DAA.

The functional subtasks in the DAA differ in numbers of rules, frequency of use, and type of knowledge. Table 14 shows that the service functions and global allocation subtasks contain over half of the total rules and half the rules that extend the partial design. Furthermore, each of the remaining three subtasks contains about fifteen percent of the total rules and fifteen percent of the rules that extend the partial design. The most frequently used design rules are the port and link management rules, which follows logically because these are the rules that create the data path. These rules are discussed in the following section.

5.1 Service Functions

An apprentice designer is first taught the tools to maintain or express a design. Along with simple bookkeeping, the designer is also taught to use the best or least expensive component. After this, various strategies of partitioning the problem into subtasks and managing the subtasks are acquired. These tools and skills, which are encompassed in the DAA as 88 rules to manage control, modules, ports, and links, are discussed in this section. An example of a rule to manage links is provided in Figure 23.

5.1.1 Control management. Like the designer dividing the task of design into subtasks, control management looks at ordering and selecting goals to complete a design. Control management involves two different types of rules. One creates goals to be later accomplished by the DAA; the other steps through outstanding goals, marks what has been done and focuses attention on what is left to do. Rules in the first category establish the major goals and subtasks described in this chapter. They recognize requests to do global allocation, VT allocation, SCS allocation, and global improvements. They also set up subgoals to perform the tasks that are listed as subheadings in this chapter (See Figure 6).

5.1.3 Port management. The last two types of management, port and link, are closely coupled and shown in Table 14 to be heavily used. Port management deals with all features of ports, while link management deals with all features of links. They are closely related because many of the features of a port are defined by the link that connects to it. This section of rules encompasses much knowledge about making the least expensive connection from one port to another. Like the designer, they consider issues of existing data paths, expanding data paths, creating new data paths and multiplexers. A designer at INTEL said:

"Reducing the number and length of connections is the single most important task in choosing an implementation."

Port management requires knowledge about creating new ports, increasing the size of old ports, creating multiplexers where needed, and rerouting connections through less expensive ports where possible. These rules are all activated by a request to *make-link*. If the source or destination port does not exist, one is created. If a second connection to an input port is requested, a multiplexer is created, or an existing one makes the connection. The only exception to this occurs when connections to the controller are requested; here just another input port is added to the controller. Wherever possible, idle existing ports are used. If a link is requested whose width exceeds the width of the port, the port and the module connected to the port are increased in width. If the expanded module is a multiplexer, then the module connected to the output of the multiplexer is also expanded. Lastly, whenever a request is made to link that could be satisfied by using an existing port or slightly expanding an existing port, the *make-link* request is modified to use this less expensive connection. Consider an example of this rerouting: a link from module A port A.out to multiplexer module B port B.in1 using bits four to one is requested, and there is

already a link from A.out to module B port B.in0 using bits eight to two. The least expensive path is to modify the link request to use B.in0 by expanding its port by one bit.

3.1.4 Link management. The last management function involves links and the expanding of existing links. This function works with the designer's knowledge codified in the previous section to make the least expensive connections between two ports.

IF:
 the most current active context is to create a link
 and no other link can be used to make the connection
THEN:
 create the link
 and mark the vt references for the source and destination ports

Figure 23. MAKE-LINK

These rules are also activated by the *make-link* goal showing the even closer coupling of this service with port management service. The link rules have knowledge about creating new links, using existing links, and expanding existing links. They are also responsible for indicating the control step when the link must be turned on to create a functioning design. They do this by informing VTDRIVE of the source and destination VT references¹⁹ responsible for the creation of this link. Given that control steps can be assigned to links, then control steps can also be assigned to modules by looking at their input and output ports and the links connected to them. This information creates the control specification part of the SCS language described in Chapter 4. This also serves to create a history, which can be traced back to the algorithmic description, of why a module, port or link exists. The knowledge of the rules in this section involves looking at existing links between the source and destination ports and either making a new link, using an existing link, or expanding an existing link to fulfill the *make-link* goal.

69

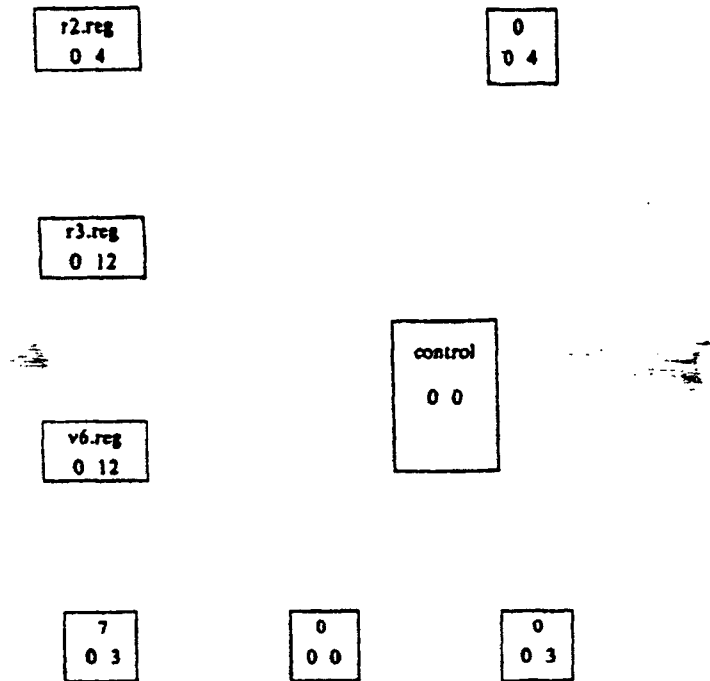


Figure 25. STATE OF DESIGN AFTER GLOBAL ALLOCATION

constants, and the large rectangle is the controller. The registers *r2.reg*, *r3.reg* and *v6.reg* are allocated because of the ISPS definitions for *cpage*, *i*, and *eadd*, respectively. A controller and four constants, one for each of the constants shown in Figure 19, are also allocated. Although unseen in the figure, the technology database and constraints are initialized as given in Tables 9, 10, 11, and Appendix B.

This section overviews allocating memories, registers, constants, controller, and database by recognizing certain features in the VT representation. These rules, like all the other rules in this chapter, use the service function rules to do their bookkeeping. An

example of a rule to allocate registers is provided in Figure 26.

5.2.1 Memories. Memory modules are allocated for VT-bodies that have word-left or word-right specified. For each memory module that is allocated, input, output, and address ports are also allocated.

5.2.2 Registers. Register modules are allocated for VT-bodies that do not have word-left or word-right attributes specified, but have bit-left or bit-right attributes.

IF:

the most current active context is declared variable allocation
and there is a vtbody that is either a carrier, vtbody or section list
and it is not an array
and it has a non-zero width
and the parent vtbody is a section list

THEN:

create a register module
and create an output port
and create an input port

Figure 26. FIND-VTBODIES-FOR-REGISTERS

For each register module that is allocated, input and output ports are also allocated. If the parent VT-body of the VT-body causing a register to be allocated is not a section list, then the register module is marked as a formal parameter and will later be removed. It will be removed because the convention used in the DAA forces all values into stable registers for the CALL and ENTER VT operations. Thus, two stable registers are unnecessarily connected.

5.2.3 Constants. Constant modules are allocated for outsnodes with a constant type attribute. For each constant module only an output port is allocated.

5.2.4 Controller. Next a controller module is created for the design. The DAA currently has only rules that create single controller designs.

5.2 Global Allocation

From the interviews in Chapter 3, the DAA was taught that global non-changing hardware is the first thing designers allocate. This partitioning makes the designer's job easier. The designers identified the base-variable storage elements, the database, the constraints, and the controller as global objects that probably would not change and should be allocated first. The base-variable storage elements are memories, registers declared globally across the design, formal parameters, and constraints.

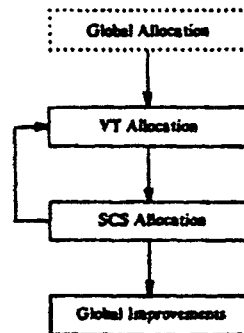


Figure 24. GLOBAL ALLOCATION

This section comprises 78 rules and is done under the context of *declared-variable-allocation*. Figure 24 shows this set of rules is fired only for the first VT-body allocated. It sets up all the modules and ports expected throughout the rest of the design and provides default constraints and timing information where none is supplied. The default constraints are given in Tables 9, 10, and 11.

Figure 25 shows the design of the decoding loop of Figures 18 and 19 after global allocation has been done. Each of the symbols represents a module with the bit width given as the bottom pair of numbers. The small rectangles are registers, the small squares are

5.2.5 Technology database. Finally, any technology-sensitive database elements not read in from the *daa.l* file are created (See Figures 8 and 9). The default values are given in the section about the database and constraints.

5.3 Value Trace Allocation

Once the designer has allocated the global non-changing hardware, the next task is to partition the whole design into smaller blocks and select a partition for allocation. Within each partition, designers allocate clock phases, operators, registers, data paths and control logic. The design is partitioned by the CMU/DA system by using the natural boundaries of the VT-bodies. A VT-body can be chosen by the designer, or the closest unallocated VT-body to the current VT-body can be found by invoking the VTDRIVE command, *mknu*, as described in Section 5.5.1. The DAA allocates the clock phases, operators, registers, data paths and control logic in two subtasks, VT allocation and SCS allocation, which are shown in Figure 27. This allows the DAA to gather all the information about register usage in the VT allocation and then allocate registers and modules in the SCS allocation.

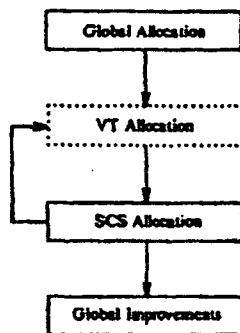


Figure 27. VT ALLOCATION

The rules used to step through the goals will either activate the next goal to be met, activate the next major task to be done, or decide no rule can perform a task (See Figure 7). Tasks that can not be performed are deleted, leaving an image of working memory in the current directory as *wm.?* for later debugging. Finally, there are rules to set up a certain goal context for every member of a list of working-memory elements. Table 14 shows that these rules compose about a fifth of the total rule firings, which follows logically from the observation that humans spend much time deciding what to do next.

5.1.2 Module management. The next type of management is module management. This set of rules has knowledge about creating, combining, and removing modules from the design. These rules allow the designer and thus other rules in the DAA, to think about creating, combining and removing modules as atomic units, rather than as individual parts. They also do the module bookkeeping discussed in the previous section. The create-module rules are activated by the goal *make-module*. If the module does not already exist, they create a module in working memory and write out a command to a collection of bookkeeping and service routines for the VT and SCS representations, VTDRIVE,⁷⁹ informing it that a module has been created. The move-module rules are activated by the goal *module-mv*. This goal is set when it is decided to make one module contain all the attributes and connections of another module, thus combining the modules. The attributes, ports, and links of one module are moved to another module, removing the first module. Lastly, the remove-module rules are activated by a goal to *module-rm*. This is done when a module is no longer needed in the design. Its links, ports, bookkeeping information, and finally the module itself are removed from the design.

72

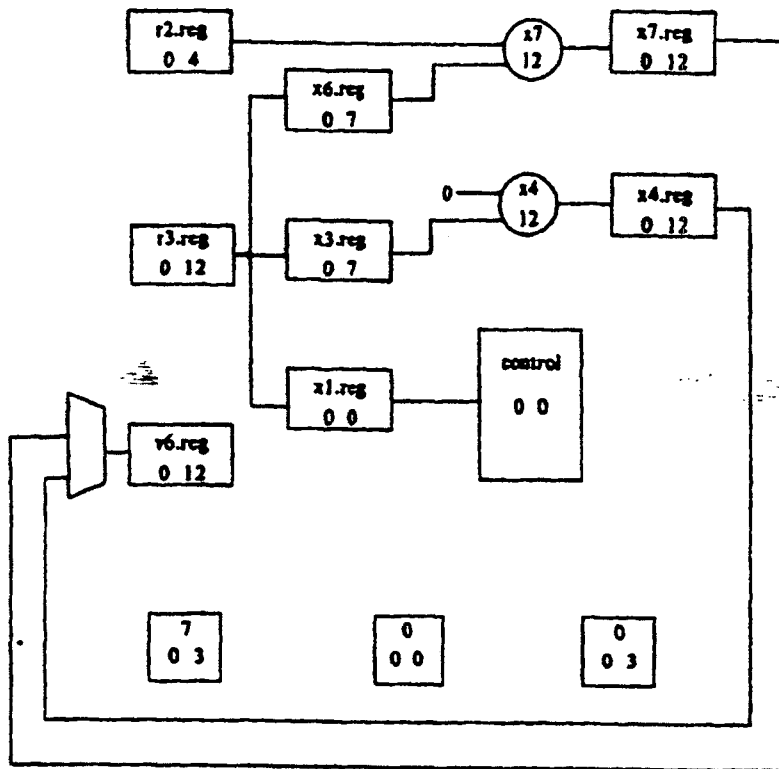


Figure 28. STATE OF DESIGN AFTER VALUE TRACE ALLOCATION

Figure 28 shows the design of the decoding loop of Figures 18 and 19 after VT allocation has been done. Each of the symbols represents a module. The circles are single function wiring modules to bring together or concatenate two sets of signals, the trapezoid is a multiplexer that gates one of its two inputs to its output, the small rectangles are registers, the small squares are constants, the large rectangle is the controller, and each of the lines represent a link between the modules. Where the links join with the modules, a port is defined. The state of the design shows the creation of the temporary registers,

(x1.reg, x3.reg, x4.reg, x6.reg and x7.reg), the concatenation modules, (x7 and x4), a multiplexer, and many connections. Along with each connection the operator assignment to control steps and the VT references are specified, so that a control specification like Figure 21 can be generated. The temporary registers are created to latch values for testability reasons. The concatenation modules are created, rather than assigned as register attributes, because they are wiring instructions and cost nothing to create. The multiplexer is created because the output of either x7 or x4 needs to be stored in v6.reg. Finally a constant has been redrawn as an input to x4.

The 55 VT allocation rules in this section assign operators to control phases, determine the minimum size needed to represent an operation, allocate temporary registers, and then associate VT operations to register modules or create ALU modules. This creation of register and ALU modules is not allocating hardware, but just mapping the VT representation into the uniform notation of the SCS representation. Rules described in the SCS section make the decision about allocating to hardware. An example of a rule to allocate temporary registers is provided in Figure 29.

5.3.1 Initialization. To reduce design-problem complexity, designers work with only one partition of a design and a listing of the currently allocated hardware. The DAA keeps an abbreviated form of the VT-bodies that are not currently being allocated in working memory, the full form of the VT-body being allocated, and the list of allocated hardware. This abbreviated form consists of only the *vtbody* working-memory structure described in Table 6. The DAA only populates working memory with the full form of the current VT-body and the input and output lists for the VT-bodies that are CALLED, ENTERED, LEFT, or RESTARTED from the current VT-body. To accomplish this there are rules to remove the abbreviated form of the VT-body and allow it to be replaced with the full form when

necessary. These rules are activated by a goal *clean*, and will remove all VT-bodies not needed in working memory.

5.3.2 Operator assignment to control steps. After the unnecessary working-memory elements have been removed and the full VT-body is loaded into working memory, the next set of activated rules groups the VT operators into control steps. This is like the designer assigning clock steps for a maximally parallel design. If the control steps have not already been preallocated by VTDRIVE using the EMUCS' algorithm, the DAA will create the maximally parallel non-pipeline design. The rules step through the VT operators sequentially, summing the delays specified for them in the technology-sensitive database. When the sum becomes greater than the maximum allowed by the working-memory element *max-delay-per-control-step*, or the delay specified in the database is -1 , a new full-control step is started. Stepping through the operators in order guarantees adherence to the data dependencies of the VT representation. The -1 delay value aligns memory, I/O, and changing context to a new VT-body operations to control-step boundaries. This synchronous and testable control structure allows LSSD techniques to be used by a layout program. Thus, these rules adhere to the maximum execution per control step, data dependency, and control dependency constraints necessary for a testable and correct design.

5.3.3 Operator trimming. The VT compiler takes a purist view of addition, subtraction and multiplication in the sense that arithmetic carries are always produced and then bit-read away when not needed. Now, this view may not seem bad, but if you want to add two 16-bit quantities and store the result in a 16-bit quantity you don't really need a 16-bit ALU with a carry. Even worse, if you have a chain of them, each will add bits to the width of the necessary ALUs. For example, 19-bit ALUs with carries could be created to add five 16-bit quantities and store them in a 16-bit quantity. DAA has two different types of rules

for operator trimming. One type of rule looks for outputs to trim (See Figures 12 and 13), while the other trims inputs. These rules and the rules for temporary registers are activated by the goal of *temporary-variable-allocation*. Outputs of addition, subtraction, and multiplication operators are trimmed to the maximum size needed by all the places the output is fed. If an operator's output has been trimmed, the inputs of the operator are trimmed. The rules for trimming inputs are careful to trim only outnodes that are produced from other operations. Other types of outnodes are trimmed by creating a dummy outnode of the trimmed size and linking it to the trimmed operator. By trimming inputs that are the outputs of other operators, whole chains of operations are iteratively reduced to their minimum size.

5.3.4 Temporary registers. Designers at IBM stressed testable designs using a LSSD testing technique. To accomplish this it is necessary to place the results of all operations that cross clock boundaries into registers. The DAA allocates temporary-register modules for outnodes that are produced by VT operators regardless of control boundaries.

IF:
 the most current active context is temporary variable allocation
 and there is a produced value outnode
 and the outnode is not associated with an architectural register
THEN:
 create a temporary register module
 and create an output port
 and create an input port

Figure 29. FIND-OUTNODES-FOR-TEMPORARIES

Each register module is allocated input and output ports. Temporary-register allocation is activated by the same goal as operator trimming, *temporary-variable-allocation*, and fires after the produced outnodes are trimmed to the correct size. Register for operators in the same control cycle are removed by the register stability rules in Section 5.4.1.

5.3.5 Register functions. Next the designers allocate hardware operators, data paths and control logic. The DAA allocates operators either as register or ALU module attributes in the SCS representation. Both types are activated by the goal *operator-allocation*. In either case, the DAA will establish the data paths by the best port and link combination and mark the necessary VT references for the control logic.

This section discusses the rules particular to register modules, whereas the next section discusses ALU modules. A designer at AT&T Bell Laboratories pointed out that it is much less expensive to build a register that can increment/decrement itself than to build an ALU which adds/subtracts a register and the constant one and stores the result back in the same register (See Figure 4). Thus, there is a cost advantage to transforming operations that act on one register into register module attributes. This is also true for division, modulus, and multiplication by powers of two or shifting by a constant amount. As register attributes, these operations are wiring and control operations.

5.3.6 ALU functions. VT operations not allocated by the previous section are allocated as ALU module attributes. This allocation is not a real allocation to hardware, but a convenient mapping of the VT representation into the same notation as all the above hardware allocation. This makes it easier for the rules in the next section to combine these operations into ALUs. The types of operations handled are single source, dual source, branch, VT-body, controller, padding, and reading/writing registers/memories operations. The rules are activated by the same goal as in the previous section, *operator-allocation*, and constitute about a tenth of the total rule firings described in Table 14.

Addition, subtraction, division, modulus, multiplication, and shifting VT operators that are not allocated by the previous section are done in this section. The single and dual input

operators addition, AND, concatenation, division, EQL, EQV, GEQ, GTR, LEQ, LSS, modulus, multiplication, NEQ, NOT, OR, SLO, SLI, SLD, SLI, SLR, SRO, SRI, SRD, SRI, SRR, subtraction, TST, XOR, and unary minus are transformed into goals of making an ALU module with the correct attribute and making links to the input and output ports. The modules, ports, and links are allocated by the rules discussed in the service function section of this chapter.

The DIVERGE and SELECT branch operators are transformed into goals of making a link from the control input to the controller and making links from each of the branch inputs to the corresponding DIVERGE or SELECT outputs.

The CALL, ENTER, and RESTART VT-body operators are transformed into goals of making links for each operator input to the corresponding input of the VT-body to be CALLED, ENTERED, or RESTARTED. This is half of the VT-body linking knowledge. The second half transforms the LEAVE operator into goals of making links for each of its inputs to the corresponding output of the VT-body to be left. These two sets of rules have forced all values to be stable in registers over VT-body context switches. This aids in creating testable designs.

The DELAY, WAIT, and DWAIT controller operators are transformed into goals of making links to the controller.

The PAD0 and PADS operators are either transformed into goals of making a module with the correct attribute and making links to and from the input and output ports, or a goal of making a link from the input to the output register. This is required because in trimming operators to their minimum size, many pad operators become unnecessary.

The last group of rules to transform VT operators involves the reading and writing of parts of registers and memories. The BIT-R and GBIT-R operators are transformed into a goal of making a link from the input register with a given displacement to the output register. The BIT-W and GBIT-W operators are transformed into goals of making a link from the input register to the output register with a given displacement and making a link from the old-value register to the output register. The link from the old-value register supplies bits to the output register not otherwise supplied by the input register. The WORD-R and GWORD-R operators are transformed into goals of making a link from the word-offset register to the address port of the memory and the output port of the memory to the output register. The WORD-W and GWORD-W operators are transformed into goals of making links from the input register to the input port of the memory with a given displacement, from the old-value register to the input port of the memory, and from the word-offset register to the address port of the memory. The link from the old-value register supplies bits to the input port of the memory not otherwise supplied by the input register.

5.4 SCS Allocation

Once the designer has created control steps, operators, registers, data paths and control logic, the design is carefully examined for possible local improvements. Designers have identified removing stable registers, combining the remaining registers, removing modules that can be replaced by using the output of another module, and combining the remaining modules as possible local improvements. The knowledge in this section uses the VT representation, the constraints, and the SCS representation built by this and the previous VT allocation to make these decisions. Figure 30 shows the relationship of this task to the other tasks in the DAA.

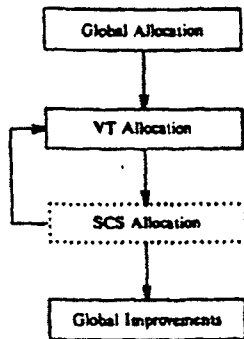


Figure 30. SCS ALLOCATION

Figure 31 shows the design of the decoding loop of Figures 18 and 19 after SCS allocation has been done. The state of the design shows the removal of the temporary registers, x1.reg, x3.reg, x4.reg, x6.reg and x7.reg, and many connections. For this simple design, it turned out that all the registers are stable and not needed to keep the design testable. Thus, they were not made permanent. Also, the two wiring operations can not be combined because they do not have the same inputs. Had the operators not been wiring operators, they would have been examined for combination using the partition and cost estimators in Sections 5.5.1 and 5.5.2 and possibly combined.

Like the designers, the 47 rules in this section remove stable temporary registers, bind other registers to existing registers or allocate new registers, remove stable ALU modules, bind other ALU modules to existing ALUs or create new ALUs, and remove unnecessary working-memory elements. These rules are activated by the goal *fold-allocation*, except for the cleanup rules that are activated by the goal *cleanup*. An example of a rule to remove stable registers is provided in Figure 32.

80

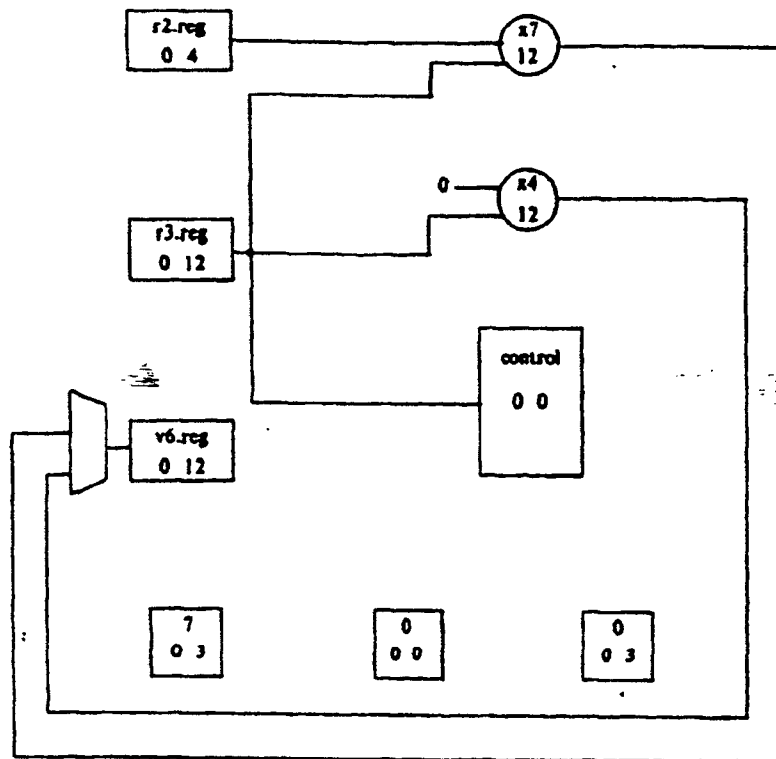


Figure 31. STATE OF DESIGN AFTER SCS ALLOCATION

5.4.1 Register stability. Good hardware designers tend to create designs that are synchronous, that is, between control steps all values are latched into registers. However, a control step may be made up of several simple operations with each output feeding the next input in the chain. The DAA looks for temporary registers that may be in such a chain and removes them. To determine if a register is stable and can be removed, the DAA asks VTDRIVE if the control step for the input port of the register is the same as the control step for the output port. It also removes registers that are only fed from a constant, because

IF:
 the most current active context is fold allocation
 and there is a temporary register
 and there is a link to the temporary register
 and there is not a link from the temporary register
THEN:
 remove the temporary register
 and remove its link

Figure 32. FOLD-NO-OUTPUT-REGISTER

these values are always available and do not need to be latched. Lastly, it removes registers that either have no outputs (See Figure 17) or whose only output feeds the controller. To remove a register a goal is created to link from the input to the output of the register and to *module-rm* the register.

5.4.2 Register allocating. Designers are always looking to share registers efficiently. Efficient register sharing not only means finding a register of the correct size functionality and connectivity, but also one that will lay out well in later stages of the design synthesis. Estimators for cost and partitioning, which have been identified through the interviews in Chapter 3, are described in Sections 5.5.1 and 5.5.2. If a temporary-register module is not removed by the DAA in the previous section, then it must be combined with existing hardware or new hardware must be created. The DAA compares the temporary register with registers that have been bound to hardware modules by size, cost estimation, and proximity estimation (See Figure 16). It compares the temporary register to all bound registers of the same size and then to all bound registers of a larger size. Next, it compares the temporary register to all bound registers of a smaller size. If the cost estimation or the proximity estimation for the comparison is greater than the *cost* or *prox* attribute of the fold working-memory element for type REGISTER, then a goal of *module-mv* is created to combine the ports and attributes for the two register modules. Otherwise, the temporary

register is bound to a new hardware register.

5.4.3 Module stability. Good hardware designers use the ability of a module to drive the output to several modules when appropriate. The DAA looks for temporary pad, concatenation, and single bit operators that are using the same inputs and attributes as other bound modules and removes the temporary ALU module. To remove a temporary module a *make-link* goal to move the output links from temporary module to the output of the bound module and a *module-rm* goal are created.

5.4.4 Module allocating. Designers are always looking to share ALUs efficiently. Efficient ALU sharing not only means finding an ALU of the correct size, functionality, and connectivity, but also one that will lay out well in later stages of the design synthesis. Estimators for cost and partitioning, which have been identified through the interviews in Chapter 3, are described in Sections 5.5.1 and 5.5.2. Designers will also combine ALUs by making tradeoffs between slower, more serial designs, and constraints on the number of modules. If a temporary ALU module is not removed by the rules in the previous section, then it must be combined with existing hardware or new hardware must be created. The DAA compares the temporary module with ALUs that have been bound to hardware modules by type, size, cost estimation, and proximity estimation (See Figure 5). It compares only modules of the same type, LOGICAL or ARITHMETIC. This is because an expert designer felt it best for testability not to combine LOGICAL operations into ALUs. Within each type it compares the temporary module to all bound modules of the same size. Then it compares the temporary module to all bound modules of a larger size. Next, it compares the temporary module to all bound modules of a smaller size. If the cost estimation or the proximity estimation for the comparison is greater than the cost or prox attribute of the fold working-memory element for type LOGICAL or ARITHMETIC, then a

goal of *module-mv* is created to combine the ports and attributes for the two ALU modules. If module attributes can be allocated, the temporary module is bound to a new hardware ALU. Otherwise, the DAA will try to compare using less stringent *cost* and *prox* values. If no match is found, the DAA will serialize the operator by moving it to the next control step and adding one to all the control steps below it. If this fails to uncover a combination, the DAA will bind the module to hardware anyway and issue a warning message.

5.4.5 Cleanup. After all the hardware has been bound, the working memory is purged of all non-abbreviated VT-bodies. These rules, which are activated by a goal of *clean*, will remove all branches, control steps, lists, operators, outnodes (See Figure 10), trees, and VT-bodies not needed in working memory. This minimizes the number of working-memory elements. After the cleanup terminates, the process repeats with the VT allocation for the next VT-body until no more VT-bodies remain. The working-memory cleanup functions account for a sixth of the total rule firings (See Table 14); without them large designs could not be done within the virtual-memory address limitations of the VAX computer.

5.5 Estimators

As the DAA evolved from the prototype system, it became clear some things could not be handled well in rules. Primarily, these were actions that required summing, counting, or checking for set membership. Represented in OPS5 rules, these calculations were terribly inefficient; they could be much better represented as algorithms in a language like C. The decision making connected with the results of these estimators was kept in rules, but the calculation was moved into a C program. Through the interviews with designers in Chapter 3, these calculations evolved to focus attention on similarities of functions and connections in VT-bodies and modules. They mimic the *back of the envelope* floor plan on which

designers base many of their decisions. These estimators have closely matched expert designer performance in partitioning and cost analysis experiments.

Primarily there are two estimators. One bridges the algorithmic to fabrication-dependent hardware-network level as a partitioner, while the other bridges the technology-independent to the technology-dependent hardware-network level as a cost function.

3.5.1 Partitioning. The first estimator is a partitioner that bridges the algorithmic to fabrication-dependent hardware-network levels. It is unlike most current partitioners of digital hardware^{80,81,82,83} because it does not require the specification of physical modules and their interconnections. That is, in our terms, it uses only the abstract and imprecise information contained in the ISPS description to make predictions of how a data path may be laid out. This layout guess is often referred to by designers as the *floor plan* of a chip. The floor plan partitioner attempts to share hardware effectively and minimize the interconnections between partitions.[†]

To share hardware operators effectively, the types of functions to be performed, the data path width, and the possibility of parallelism must be considered. To minimize interconnects, the common use of carriers and the direct passing of a value from one function to the next must be taken into account. In partitioning, all the above factors must be taken into account with the style of implementation and technology-sensitive information. They must be quantified, weighted, and combined to give a measure of what functions are similar and belong together, and what functions are not similar and can therefore be

[†] This work was carried out under my supervision at AT&T Bell Laboratories in the summer of 1982 by Michael McFarland S. J.

separated without great cost. A number of effective mathematical techniques have been developed to estimate similarities or distances between data points. One method for measuring the similarity of two objects⁸⁴ involves counting the number of properties they have in common. Suppose that two objects, 1 and 2, are to be compared. If A_1 is the set of properties belonging to object 1 and A_2 is the set of properties belonging to object 2, then an estimate of the similarity of the two objects can be based on the similarity of A_1 and A_2 . A common measure of similarity is the so-called Jaccard, which is equal to

$$\frac{|A_1 \cap A_2|}{|A_1 \cup A_2|}$$

where for any set A , $|A|$ is the cardinality or the number of elements in A .

The measure of similarity developed here is analogous to the Jaccard concept described above. Instead of common properties, the algorithm counts the number of components and connections that could be shared by two functions, where components is an abstract category that can be made to correspond to gates or other hardware modules, depending on the implementation style. For each pair of functions in an ISPS description, two similarities are calculated. One, called the operator proximity, measures the commonality in the operations performed by the two functions. The other, called the register proximity, is a measure of the number of common register references and interconnections as a percentage of the total number of register references and interconnections used by the two functions. The complete similarity measure is then a weighted average of the operator proximity and register proximity. The way this average is formed and the weights used can be controlled by user-settable parameters.

The operator proximity between functions f_1 and f_2 is the cost of the sharable hardware divided by the cost of all the hardware needed to do all the computations in f_1 .

and f_2 separately.

The cost of a function f is the number of components needed to implement all the computational operators, such as adds, shifts, comparisons, and so on, used in the body of the function. The cost is based on the individual operators and the extent to which they can share the same hardware. For each type of operator i , there is a user-defined technology-sensitive cost c_i , which corresponds to the number of gates or components needed to implement the function, or some other measure of cost. For each pair of operator types, i and j , there is an overlap cost o_{ij} , which represents the amount of hardware that could be shared between the two operators. For example, if an adder takes six gates per bit and a subtractor eight gates, six of which can be shared with an adder, $c_{add} = 6$, $c_{sub} = 8$, and $o_{add,sub} = 6$.

For a set of operators S , the function $\text{cost}(S)$ processes the operators in S one by one, finding the cost of adding the hardware capability for that operator to the hardware for the operators already processed. The cost of adding an operator to a given set is found by computing the cost of combining the operator with each operator in the set and then taking the minimum of those costs. The cost of combining operators i and j is $c_i + c_j - o_{ij}$.

The operator proximity between function f_1 , with operators S_1 , and f_2 with operators S_2 is then

$$\frac{\text{cost}(S_1) + \text{cost}(S_2) - \text{cost}(S_1 \cup S_2)}{\text{cost}(S_1 \cup S_2)}$$

because for any sets S and T , $S \cap T = S + T - S \cup T$. If f_1 and f_2 can be carried out simultaneously, then the operator proximity is lowered by a technology-sensitive factor that is set by the user.

The register proximity between functions f_1 and f_2 is the number of bits of carrier referenced in common, and of common interconnections, divided by the sum of the bits of carrier references and interconnections used in either one. Carriers in the ISPS description are weighted differently depending on what they stand for and how they are used. It is assumed that globals and arrays will remain as they are in the final design, so they are given full weight. Registers in which a value is directly passed between functions are also assigned full weight, because there must be some interconnection, whether the value is in that particular register or not. Non-architectural register references are downgraded by a technology-sensitive factor set by the user.

The total proximity is a weighted average of the operator proximity and the register proximity. This weighting can be done in two different ways. Static weighting simply uses two user-settable technology-sensitive weighting factors. Dynamic weighting takes into account the relative importance of operators and registers in the individual functions. Thus, if the functions are computation intensive, the operator proximity would weigh more heavily, whereas if they simply involve moving values around, the register proximity would weigh more heavily.

The partitioner has closely matched the performance of five expert designers participating in a partitioning experiment for the MCS6502 description.⁸⁵ The information is used by the DAA as a high-level floor plan to aid in making decisions about whether to create new modules or upgrade existing modules to contain the required connections and functionality. Further information is given in the discussion of register and module allocating in Sections 5.4.2 and 5.4.4.

5.5.2 *Cost estimator.* The second estimator is a hardware pricer that bridges the technology-independent to technology-dependent hardware-network levels. It is unlike the partition estimator because it requires that the system be specified by technology-independent modules and their interconnections. It provides information about what percentage of the required hardware for a new module already exists in another module. That is, in our terms, it uses only the abstract and imprecise information contained in the SCS description to make predictions of how much it would cost to upgrade the functions and interconnections of an existing module to contain a new module. Whereas the partition estimator gives a high-level floor plan, this estimator gives a much more local view, thus augmenting the high-level floor plan with more detailed information.

To price hardware modules, the types of functions to be performed, the data path width and the possibility of parallelism must be taken into account, along with the input and output data path connections. In pricing, all the above factors together with the style of implementation and technology-sensitive information must be considered. They must be quantified, weighted, and combined to give a measure of how expensive a new module would be as opposed to the cost of sharing an already existing module. One method for measuring the cost of adding a new object to an old object involves calculating the percentage of the new object's properties that already exists in the old object. Suppose that two objects, numbered 1 and 2, are to be compared. If A_1 is the set of properties belonging to object 1 and A_2 is the set of properties belonging to object 2, then an estimate of the increased cost of object 2, if it is to include object 1, can be based on the intersection of A_1 and A_2 scaled by A_1 . This cost function is equal to

$$\frac{|A_1 \cap A_2|}{|A_1|}$$

The cost estimator developed here is analogous to the concept described above. For each pair of modules in a SCS description, two costs are calculated. One, called the module cost, measures the percentage of operations performed by the module 1 not in module 2. The other, called the port cost, measures the percentage of interconnections used by module 1 not in module 2. The complete cost measure is then a weighted average of the module cost and port cost.

The module cost between modules m_1 and m_2 is defined to be the cost of the hardware that can be used to do computations in both m_1 and m_2 , that is, the sharable hardware, divided by the cost of the hardware needed to do the computations in m_1 .

The cost of a module m is defined in the same way as the cost of a function f is defined in the previous section. The module cost between modules m_1 , with operators M_1 , and m_2 with operators M_2 is then

$$1 - \frac{\text{cost}(M_1 - M_2)}{\text{cost}(M_1)}$$

because for any sets S and T , $S \cap T = S - (S - T)$, where $S - T$ is the relative set difference defined by

$$A - B = \{x \mid x \in A, x \notin B\}$$

If m_1 and m_2 are used in parallel, then the operator cost is -1.

The port cost between modules m_1 and m_2 is the cost of the hardware that can be used to do connections in both m_1 and m_2 that is, the sharable hardware, divided by the

cost of the hardware needed to do the connections is m_1 . The cost of a module m is the number of components needed to implement all the connections used in the module. The cost is based on the individual ports and the extent to which they can share the same hardware.

For a set of ports P , the connection cost(P) processes the ports in P one by one, finding the cost of adding the hardware capability for that port to the hardware for the ports already processed. When port references are made to wiring modules such as MUX, DEMUX, BUS, ~~CONCAT~~, PAD0, or PADS, the ports of these modules are recursively taken into account.

The module cost between modules m_1 , with ports P_1 , and m_2 with ports P_2 is then

$$1 - \frac{\text{cost}(P_1 - P_2)}{\text{cost}(P_1)}$$

The total proximity is a weighted average of the module cost and the port cost. This weighting is done dynamically, taking into account the relative importance of functions and ports in the individual modules. It is given by:

$$\frac{\text{cost}(M_1) - \text{cost}(M_1 - M_2) + \text{cost}(P_1) - \text{cost}(P_1 - P_2)}{\text{cost}(M_1) + \text{cost}(P_1)}$$

Thus, if the modules are computation intensive, the module cost would weigh more heavily, whereas if they simply involve connections, the port cost would weigh more heavily.

The pricer has closely matched the performance of nine expert designers participating in a pricing experiment for the MCS6302 description.⁶⁶ The information is used by the DAA as a high-level floor plan to aid in making decisions about creating new modules or upgrading existing modules to contain the required connections and functionality. Further

information is given in the discussion of register and module allocating in Sections 5.4.2 and 5.4.4.

5.6 Global Improvements

As a design nears completion, the designers start examining it for things that are no longer needed or could be better shared.

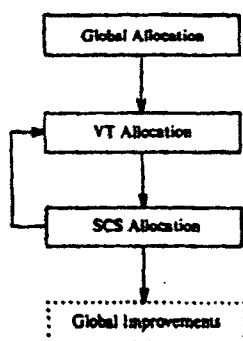


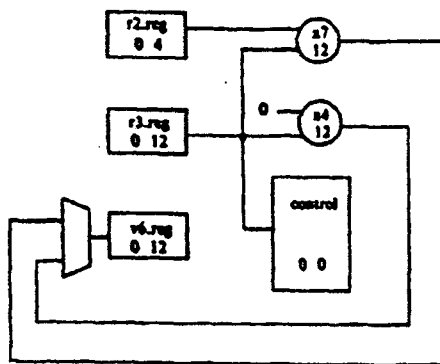
Figure 33. GLOBAL IMPROVEMENTS

This cleanup includes unused modules and ports, stable registers, duplicated multiplexer input ports, and modules that could be eliminated through use of common outputs. Finally, the designers examine the design for use of buses in high traffic areas. This set of 46 rules, like the global allocation set of rules described above and shown in Figure 33, is activated only once per design. Once these rules have fired, the design is done and no more VT-bodies can be processed without incurring errors.

Figure 34 shows the design of the decoding loop of Figures 18 and 19 after global improvements have been made. The state of the design shows the removal of the unused constants and the input ports to r2.reg and r3.reg. Because the initial ISPS was so simple,

no multiplexer cleanup or bus allocation is required.

The DAA's global improvements remove unreferenced modules, remove unused registers, reduce multiplexer trees, use fan-out from modules, and allocate bus structures. An example of a rule to allocate bus structure is provided in Figure 35.



: Figure 34. STATE OF DESIGN AFTER GLOBAL IMPROVEMENTS

3.6.1 *Unreferenced.* Unnecessary modules and ports are created during the design process because some constants defined in the VT representation are not really needed in the hardware design; they only serve as offsets to wiring operations. Also the initial ISPS description may define carriers that are not really used. The DAA also tends to create a few extra multiplexer and bus ports in the act of finding the least expensive routing port. The first step towards cleaning up the design is the removal of unreferenced ports, modules, and formal parameter registers. These rules are activated by the goal *unreferenced*. If a port does not have a link connected to it, it is removed. If all the ports of a module are removed, the module is removed. If the module is a BUS or MUX type module, its bookkeeping information is also removed. Finally, formal-parameter register modules are

removed, moving all their output links to the multiplexer or register that is feeding it. This is done because, as explained above, all values are stable on entry to VT-bodies. Thus, these registers are stable and are not needed to latch the input values.

5.6.2 Multiplexer cleanup. The removal of the formal-parameter registers tends to create trees of multiplexers, sometimes with duplicated input ports. The DAA has rules for the flattening out of the multiplexer tree structures and the removal of the duplicated input ports. It also removes multiplexers without output links and replaces multiplexers or buses with only one input port with links. These rules are active for the same goal as above, *unreferenced*.

5.6.3 Multiple fan-out cleanup. The DAA takes one more look for pad, concatenation, logical, and single bit modules that are using the same inputs and attributes as other bound modules to replace them with the ability of a module to drive output to several modules. It has to do this a second time, because the unreferenced and multiplexer cleanup can create more fan-out sharing.

5.6.4 Bus utilization. Now that everything has settled down, the DAA examines the design for possible bus usage.

IF:

the most current active context is bus allocation
and there is a module that is a multiplexer
and there is another module that is also a multiplexer
and there is a link from a non-bus module to the first multiplexer
and there is a link from that module to the second multiplexer
and there is a link from another non-bus module to the first multiplexer
and there is a link from that module to the second multiplexer

THEN:

place these connections on an idle bus

Figure 35. CONVERT-MUX-INPUTS-TO-BUS

Because buses are expensive in area and in power, they are used sparingly in VLSI design. Thus, the DAA tends to use buses only if there is a cost gain over using discrete multiplexers (See Figure 3). There are three types of bus rules, all activated by the goal *bus-allocation*. There are rules to propose that multiplexer inputs be moved to a bus, there are rules to find the best bus to place inputs on, and finally there are rules to add other inputs and outputs to a bus. Figure 36 shows an example of a multiplexer configuration on the left that would be allocated as the bus configuration on the right after these rules fire. Moving multiplexer inputs to a bus is proposed if there are two multiplexers, and on each multiplexer there are two identical inputs. Once this proposition has been made, buses containing both inputs are checked first, followed by buses containing only one input. The input ports of the bus and the proposed input ports are checked for timing collisions. If the bus is not idle, then this is followed by buses of the same, larger and smaller sizes.

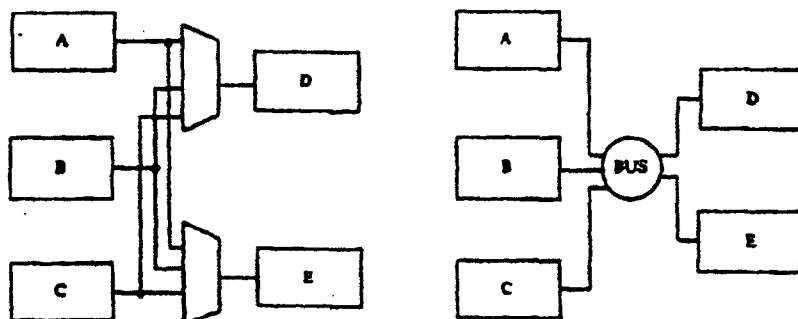


Figure 36. AN EXAMPLE BUS ALLOCATION

If none of the buses were idle and the limit for the number of buses has not been exceeded, then a new bus is allocated. Otherwise whenever an idle bus is found, the inputs and outputs are optimally linked onto the bus.

Finally, there are three configurations of multiplexers, buses and modules that will be tested for moving to bus ports. If there is a module with an output to a multiplexer and a bus, and the bus has an output to the same multiplexer, then a check is made to see if the output from the module to the multiplexer can be moved onto the bus. If there is a module with an output to a multiplexer and a bus with an output to the same multiplexer, a check is made to see if the output from the module to the multiplexer can be moved onto the bus. Lastly, if there is a module with an output to a multiplexer and a bus, a check is made to see if the output from the module to the multiplexer can be moved onto the bus.

5.6.5 Unreferenced and final cleanup. After the multiple fan-out cleanup and bus allocation, the *unreferenced* goal is activated one last time for a final cleanup. See the above sections on unreferenced and multiplexer cleanup for more information.

5.7 Knowledge Summary

Chapter 5 has shown how designers partition the design task into four major tasks. These tasks involve global allocation of architectural modules and registers; local allocation of control steps, operators, registers, data paths and control; global allocation of operators and registers; and global improvements to the design. Within these tasks, a dominant goal is to partition the design into smaller, more manageable pieces. While making the design more manageable, it is also important to retain a global view of the hardware. Designers retain this global view by using high-level floor plans that are filled in as the design proceeds. This floor planning mechanism is composed of estimators for partitioning and pricing hardware. These estimators are concerned with connectivity and functionality of the networks of hardware. The designers feel it is important to design testable synchronous designs. This constraint is even more important than making the least expensive

connections or optimally sharing hardware. After testability, the most important consideration is how the design will lay out or how to minimize connectivity. Thus, creating a good design is not just minimizing components, but paying careful attention to testability and connectivity.

The knowledge in the DAA system has been gathered using a KBES approach. The approach allowed the separation of expert knowledge from the reasoning mechanism. This facilitated the incremental addition of new rules and the refinement of old ones because the rules could be written to have minimal interaction with one another. The rules in the DAA system were also written with the goal of placing technology-sensitive constants in the working memory. Although these goals made it easier to add new rules and change the target technology, they did not take full advantage of the optimizations possible in the OPS5 system. The general design of the DAA system was not highly interactive because the Franz Lisp version of the OPS5 interpreter is slower than could be tolerated by the expert designers. If the DAA were to be rewritten, a more interactive environment could be attained by using either the BLISS version of OPS5, which is about 12 times faster than the Lisp version (.92 rules/second in the Lisp version versus 10.6 rules/second in the BLISS version for a major part of the MCS4502 design) or the OPS83 system. Because OPS5 is particularly bad at summing, counting, and checking set membership, to speed up the DAA the rules for partitioning and cost estimating were rewritten as C procedures. Two final burdens in using OPS5 were that the structure of working memory was flat and there was no facility for time profiling the execution of rules. A hierarchical working-memory representation would have decreased the use of variables as pointers in the rule memory and thus, decreased execution time. In writing rules, the order of the antecedents can vary the run time of the system by a factor of ten. A time profiler or the dynamic ability to change

the internal representation of the rules in the Rete network would have corrected this problem. In summary, use of the KBES approach decreased the time it took to gather and verify the knowledge used by expert VLSI designers, but use of OPS3 increased the time it took to create designs.

Chapter 6: THE IBM SYSTEM/370 EXPERIMENT

After the DAA successfully designed a MCS6502 microprocessor, it had to be determined whether the system had also acquired knowledge about processor design in general. In this regard, an experiment was designed to see whether the DAA could design a processor substantially different and more complex than the MCS6502.

An ISPS description was chosen for the complete IBM System/370 from the descriptions maintained at Carnegie-Mellon University. This description included memory-management operations, channel controller I/O instructions, and all the 370 instructions except the extended-precision floating point, the characters under mask, the edit and mark, and the packed-decimal instructions. The unmodified System/370 description, missing only a small percentage of the total 370, is more than 10 times larger than that of the MCS6502,[†] and it had not been used to build the DAA. Important benefits of this choice are that a single-chip design of the 370 had been made at IBM, information is publically available, and Claud Davis, the design team manager and a key designer, was willing to critique the design. Thus, the experiment was a fair and convenient way to test the generality of the DAA's design knowledge.

This chapter presents the design produced by DAA, D370, the design produced by Claud Davis, the IBM System/370 bipolar gate array micro-processor chip, μ 370,⁸⁷ and a comparison of the two designs. For each difference, possible changes in the CMU/DA system and the DAA are discussed.

[†] The next bigger description, the Digital Equipment Corporation VAX 11/780, wouldn't compile through the VT compiler in a six megabyte address space.

6.1 The D370 Design

The experiment was begun by taking the ISPS description of the 370 (2,841 lines / 63,307 bytes) and translating it into VT notation (47,486 lines / 1,581,231 bytes), which comprised 6,078 operators (1,268 of which are branch operators), 154 constants, 251 VT-bodies, 51 map declarations, two section lists, 9,514 values (88 of which are declared registers), 2,692 outputs from VT-bodies, 14,240 input to VT-bodies (27 of which are formal inputs declared in the ISPS description), and 63,899 inputs to operators. The VT file was used as input to VTDRIVE, where high-level partition planning was done as discussed in Section 5.5.1. Then each VT-body was given to the DAA in turn. The DAA designed the D370, its version of the System/370, in 47 hours of CPU time on a VAX 11/780 with six megabytes of memory and two memory controllers. The D370 was designed without rule modifications or design iterations of any type.

The D370 is an IBM System/370 data-flow design using a $50x$ clock, where x is some scaled unit of time like μ seconds, with multiplexer and bus style data paths. The DAA's constraints were set to produce a high-performance machine — that is, it could use as much hardware as required to allocate the data paths and retain maximum parallel operator usage. To meet this performance constraint, the D370 has eight-bit, 24-bit and 64-bit buses, 32-bit, 64-bit and 68-bit ALUs, a few discrete components, six memory arrays, and a great many architectural registers. This section discusses the functional blocks of the D370 design.

100

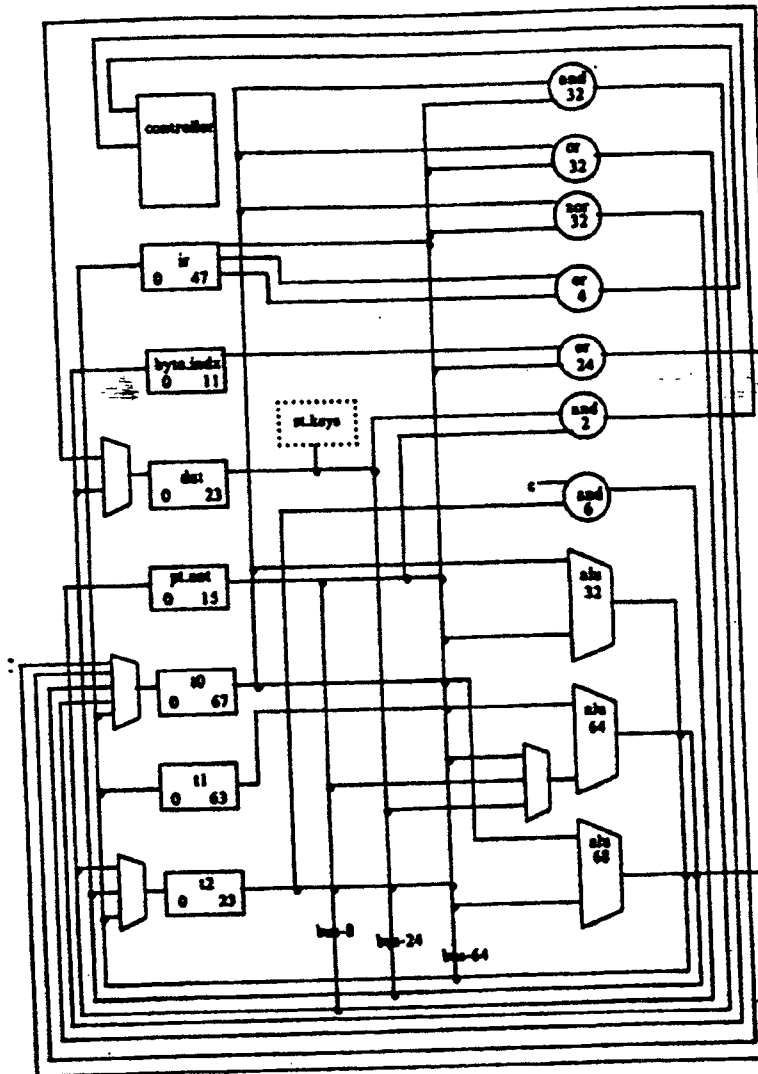


Figure 37. THE D370 DESIGN - PART 1

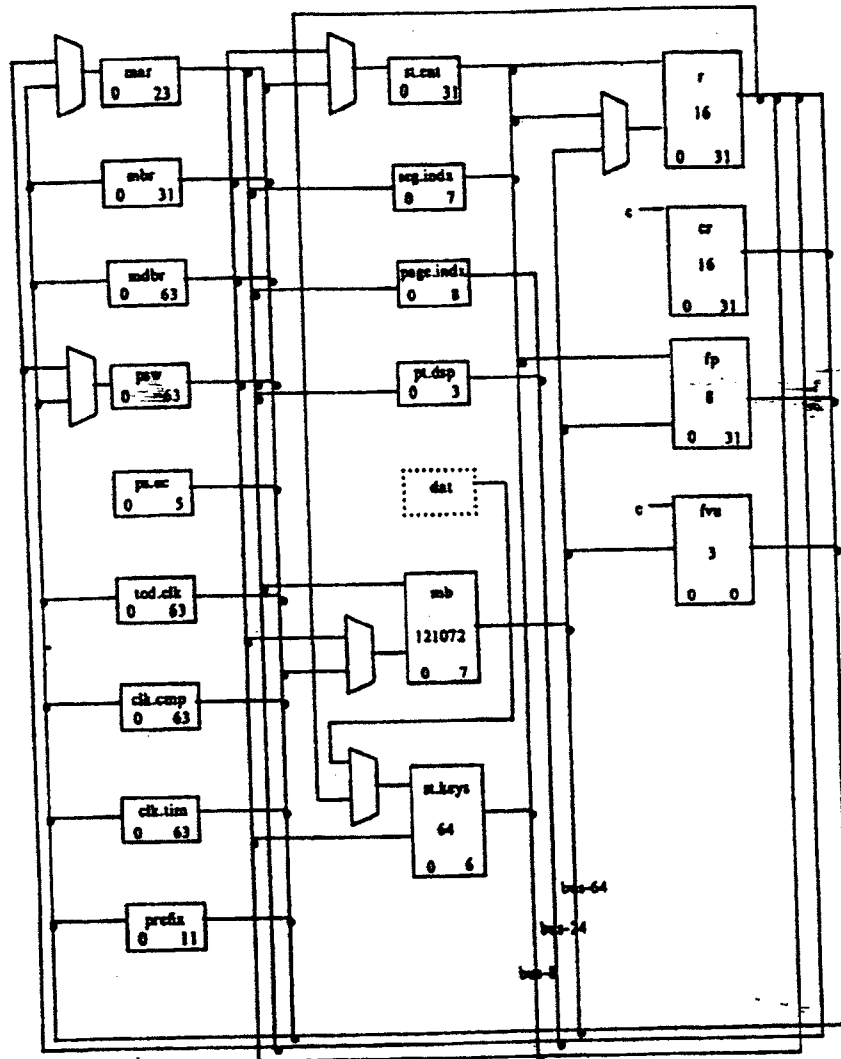


Figure 38. THE D370 DESIGN -- PART 2

6.1.1 The three ALUs. The three ALUs and bus sizes arose from three different groups of data operations and major transfer widths in the IBM System/370. The basic basing style placed a temporary register before an input to each ALU and assumed the ALU latches its result so it can be read on the next clock phase transition. Thus, a two-phase clock set up the inputs to an ALU in one clock cycle and stored the result on the next clock cycle.

For clarity, the data path of the D370 is drawn in two separate figures, Figures 37 and 38. The connection between the two figures is through the eight-bit, 24-bit and 64-bit data buses. Figure 37 contains the arithmetic portion, the temporary registers, and the controller; Figure 38 shows the architectural registers, including the register arrays, such as the 16 general purpose registers *R*.

The 32-bit ALU is used for most of the arithmetic operations in the System/370 architecture. It can ADD, SUBTRACT, and COMPARE two binary numbers from the *T0* temporary register and the 64-bit bus. It gates a result out on this bus.

The 64-bit ALU is used for most of the address calculation operations and a few low-frequency operations, such as MULTIPLY and MODULUS in the System/370 architecture. It can ADD, SUBTRACT, COMPARE, MULTIPLY, MODULUS, and SHIFT RIGHT two binary numbers from the *T1* temporary register and a bus. It can gate a result out on either the 24-bit or 64-bit bus.

The 68-bit ALU is used for most of the floating-point operations in the System/370 architecture. This ALU can ADD, SUBTRACT, COMPARE, and SHIFT LEFT two binary numbers from the *T0* register and the 64-bit bus. Its result is gated onto the 64-bit bus.

6.1.2 The discrete components. Not all data manipulation is done in the ALUs. To aid debugging, one of our expert designers from INTEL keeps single-function logic outside the ALU. Thus, the logic instructions are implemented with separate distributed logic elements: the 32-bit AND, OR, and XOR.

Smaller logic elements are provided for a variety of functions. The four-bit OR takes input from two fields of the instruction register *IR* and feeds the result to the microcontroller. This aids in instruction decoding. The virtual storage system uses three discrete components. The 24-bit OR takes input from the byte index *BYTEINDX* register and the 64-bit bus and places the result in the dynamic address translation *DAT* register. The two-bit AND takes input from the 24-bit bus and the page table entries *PT.ENT* and feeds the result to the microcontroller. The six-bit AND takes input from a group of constants and *TZ* and places the result on the eight-bit bus.

6.1.3 The memory arrays. The D370 architecturally defines the primary memory *MB*, the storage keys *ST.KEYS*, the general purpose registers *R*, the control registers *CR*, the floating point registers *FP*, and the floating point error registers *FVU*.

Table 15. MEMORY ARRAYS IN THE D370 DESIGN

Abbreviation	Bits	Words	Address	Inputs	Outputs
<i>MB</i>	8	121072	24bb	8bb, 64bb	64bb
<i>ST.KEYS</i>	7	64	<i>R</i> , <i>DAT</i>	8bb	8bb
<i>R</i>	32	16	8bb	8bb, 64bb	<i>ST.KEYS</i> , 8bb, 24bb, 64bb
<i>CR</i>	32	16	Constants		64bb
<i>FP</i>	32	8	8bb	64bb	64bb
<i>FVU</i>	1	3	Constants	64bb	64bb

Table 15 lists the bit width of each memory array, the number of words in the array, and what buses *bb* or registers connect to the address, input, and output ports. Thus, there are

64 storage keys, each seven bits wide, with their address port connected to the general purpose registers and the dynamic address translation register. The input and output ports are connected to the eight-bit bus.

6.1.4 *The architectural and temporary registers.* The D370 architecturally defines many registers. Table 16 lists the bit width of each register and tells what buses or registers connect to the input and output ports.

Table 16. REGISTERS IN THE D370 DESIGN

Abbreviation	Name	Bits	Inputs	Outputs
MAR	memory address	24	24bb, 64bb	24bb, 64bb
MBR	memory buffer	32	64bb	8bb, 64bb
MDBR	memory double buffer	64	64bb	8bb, 64bb
PSW	processor status word	64	24bb, 64bb	8bb, 24bb, 64bb
PSEC	extended code	6		64bb
TOD.CLK	clock	64	64bb	64bb
CLKCMP	clock comparator	64	64bb	64bb
CPU.TIM	CPU timer	64	64bb	64bb
PREFIX	prefix	12	64bb	64bb
IR	instruction register	48	64bb	64bb, 4-bit OR
ST.ENT	segment table entry	32	8bb, 64bb	8bb
PT.ENT	page table entry	16	8bb	8bb, 64bb, 2-bit AND
SEG.INDX	segment index	8	24bb	8bb
PAGE.INDX	page index	9	24bb	24bb
PT.DSP	page table displacement	4	24bb	24bb
BYTE.INDX	byte index	12	24bb	24-bit OR
DAT	dynamic address translation	24	24bb, 24-bit OR	24bb, ST.KEYS
T0	temporary 0	64	64bb, 32-bit AND, OR, XOR	64bb, 32-bit ALU, 68-bit ALU, 32-bit AND, OR, XOR
T1	temporary 1	64	64bb	64bb, 64-bit ALU
T2	temporary 2	24	8bb, 24bb, 64bb	8bb, 24bb, 64bb, 6-bit AND

Thus, the instruction register is 48 bits wide with its input connected to the 64-bit bus and its output connected to the 64-bit bus and the four-bit OR described above.

6.1.5 The control specification. A symbolic microcode word controls the D370. A microcode word is required for each cycle of the machine. The generation of either PLA or ROM based micro-engine is possible in later phases of the design synthesis task. A sample sequence from the dynamic address translation VT-body has the 24-bit bus gating the *MAR* to the *MB* address port, the 64-bit ALU adding the *MAR* from the 24-bit bus and the temporary register *T1* and gating the result on the 64-bit bus to the *MB* input port, with the six-bit AND anding the temporary register *T2* and a constant, gating the result on the eight-bit bus to a field in the *PT.ENT*. This illustrates the high degree of parallelism possible in the D370.

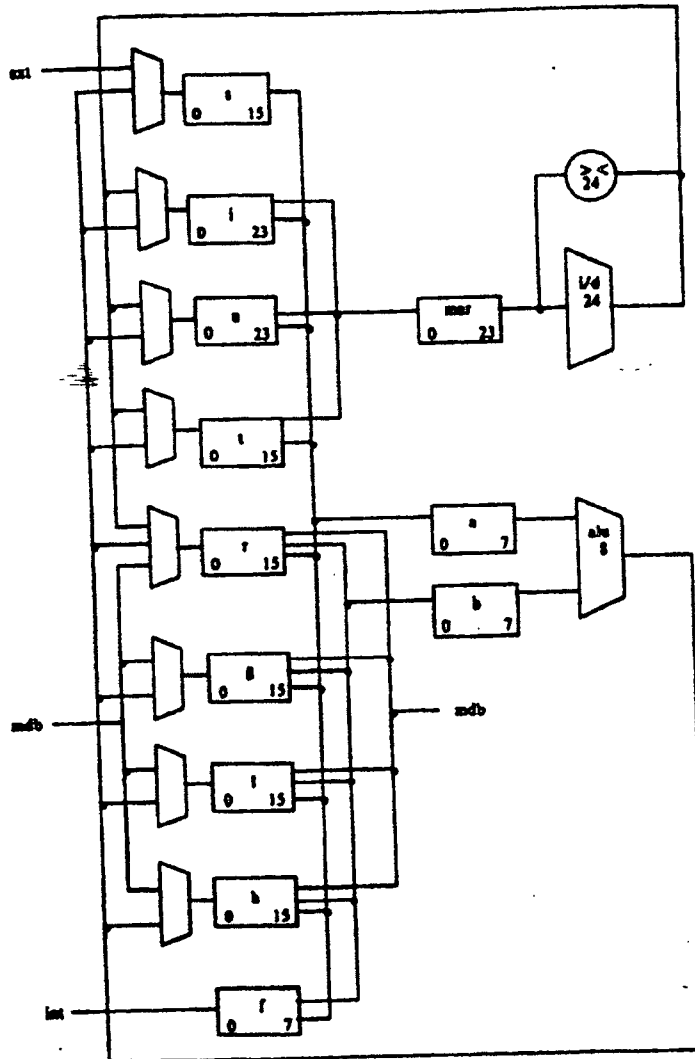
6.2 The μ 370 Design

The μ 370 is an IBM System/370 micro-processor data flow on a single bipolar gate-array masterslice chip. It uses a 100-nanosecond cycle clock and is capable of executing 200,000 instructions per second. The physical chip is 7x7 mm and dissipates 2.3 watts. The plan was to use no more 5000 wired circuits, 3 watts of power, and 200 pins. To meet these size and power constraints, the problem was divided into on-chip and off-chip sections. This section discusses the functional blocks of the μ 370.

6.2.1 The on-chip functional block. The on-chip functional block has an eight-bit ALU, a 24-bit incrementer/decrementer, I/D, a 24-bit shifter, two nine-bit parity generators, 17 eight-bit working registers, two eight-bit buffer registers, a 16-bit status register, a 24-bit register, and hardware to calculate the next microcode address. These components are wired together with two fan-in eight-bit buses, a fan-out eight-bit bus, a bidirectional 16-bit bus, a fan-in 24-bit bus and a fan-out 24-bit bus. These are shown in Figure 39.

The eight-bit ALU can ADD, SUBTRACT, OR, AND, and XOR either binary or

106

Figure 39. THE $\mu 370$ DESIGN

packed-decimal numbers. The arithmetic operations of the ALU can be controlled directly from a microcode field or indirectly through a status bit located in register *S*. This indirect control feature allows sharing of microprogramming routines for the ADD and SUBTRACT operations. Two eight-bit buses feed two eight-bit buffer registers, *A* and *B*, which feed the ALU. These two registers can selectively gate groups of four bits that correspond to hex digits within the byte or pass the complete byte to the ALU. This gating is used for decimal operations. The *A* register can also pass its eight bits, rotated by four bits to reverse its two hex digits. This is used by the pack and unpack instruction. The output of the ALU is placed on an eight-bit bus gated to all the working registers.

The 24-bit I/D is a special purpose adder that can add a 24-bit binary number with the constants 0, 1, 2, 3, -1, -2, or -3. The constant input is directly controlled from a microcode field. This I/D is dedicated to address calculations. An address is gated from a set of three working registers onto the 24-bit bus feeding the memory address register *MAR*. Any value present in this register is gated to the memory address bus MAB, the input of the I/D, and a shifter (discussed below). The output of the I/D is placed on the 24-bit fan-out bus and gated back to the same three registers that fed the *MAR*.

The 24-bit shifter can do a few complex shift operations to produce a 16-bit result. This shifter is dedicated to handling the 12-bit address field used for page addressing in the virtual storage system. Input is gated from the *MAR*, and output is gated onto the 24-bit fan-out bus.

Two nine-bit parity generators check the parity of each data byte arriving at the chip and place it on the 16-bit memory data bus, MDB. They can also affix a parity bit to each data byte leaving the chip from the MDB.

The $\mu 370$ has 17 eight-bit registers, two eight-bit ALU buffer registers, a 16-bit status register, and a 24-bit *MAR*. The 17 eight-bit registers are grouped in functional pairs and triples. The *R*, *G*, *L*, and *H* register pairs are primarily the memory data registers, MDRs. Because the *G* register pair has special microcode branching capabilities, it is the OP code register. The *I* and *U* register triples are the program counter and operand register, respectively. The *T* register pair is the local-store address. The *I*, *U*, *T*, and *R* register groups can pass two bytes of data to one another with or without a displacement. This feature is part of the virtual storage management of the IBM System/370 architecture. The *S* register pair is a CPU status and serves as input to the microcontroller. *S/* can be set and reset by external inputs. The interrupt register *F* is also acttable by external conditions.

A 34-bit microcode word controls the $\mu 370$. A microcode word is required for each cycle of the machine and is fetched during the last 75 nanoseconds of each cycle from the read only store, ROS. To select the next ROS word, a 16-bit address is generated in the first 25 nanoseconds of each cycle. Six bits of the ROS address are taken directly from the ROS word. The low order two bits are extracted from conditions within the chip. ROS fields dictate the internal conditions to be examined. The remaining bits are taken from the previous ROS address. However, if an external-trap bit is raised, the ROS address is forced to a specific value for a trap handler. Possible traps are parity errors, IPL request, page overflow, storage wrap, memory protect violation, stop request, and I/O control.

6.2.2 The off-chip functional block. The off-chip functional block has the architectural registers, two external memories, an I/O port, and the ROS. The $\mu 370$ chip uses 512 bytes of architectural registers. They are kept in a local store that can be accessed in 60 nanoseconds. The local store, which is limited to 64K bytes, is addressed by the *T0* and *T1*

registers. The μ 370 uses up to 16 megabytes of memory, which is addressed by the MAB, while data is gated on the MDB. *Read*, *write*, *memory-1*, *memory-2*, and *ready* signals allow up to two memories of any speed to be interfaced to the chip. If neither memory line is asserted, the MDB is connected to an I/O bus that uses the MAB to choose the I/O device being serviced.

6.3 The D370 and μ 370 Design Comparison

The previous two sections have discussed the individual attributes of the two designs. This section brings those designs together by comparing and contrasting their differences. Claud Davis compared the two designs at IBM Poughkeepsie. During his career of over 25 years at IBM, Davis has worked on designs and managed teams of designers for the 701, 702, 7074 MA, 360/50, FAA, 360/67, and the μ 370. His vast experience with the higher-performance processors and the μ 370 made his critique valuable in two ways. First, we could determine what is needed for a single-chip IBM System/370 architecture; second, we could determine what is needed for a higher performance processor. Davis summarized his comparison thusly:⁸⁸

"The 370 data-flow we reviewed exhibited the quality I would expect from one of our better designers. The level of detail was what we call second level design. This encompasses all 'architected' registers, status latches and sufficient working registers to implement the functions defined by the instruction set. This level of design is independent of implementing technology.

The review included a test for 'architected' registers, data path widths, latches for exceptional conditions, signs, and latches for temporary information in multi-cycle instructions.

The assumptions for clocking and controls were examined and found to be consistent."

Relevant sections of the complete transcript⁸⁹ are included in Appendix D.

Table 17. IBM SYSTEM/370 — DESIGN DIFFERENCES

Design	D370	μ 370
Objectives	High performance, technology sensitive, independent of power and I/O pins; paper design	Strict observance of technology criteria such as number of wired circuits, power, and I/O pins; working chips
ALUs	32-bit, 64-bit, and 68-bit; Binary numbers; hardware for virtual memory, floating point and multiply	Eight-bit and 24-bit; Binary and packed-decimal numbers; microcode for virtual memory, and multiply
Buses	Eight-bit, 24-bit, and 64-bit; bidirectional	Three 8-bit, a 16-bit, two 24-bit; fan-in, fan-out and bidirectional
Memories	12-byte buffer; single ported	Eight-byte buffer; single ported
Registers	Discrete	Memory array

In the following discussion, differences are grouped by objectives and functional blocks including: ALUs, buses, memories, and registers, which are summarized in Table 17. For each difference possible changes in CMU/DA and DAA are discussed.

6.3.1 The objectives. The objectives and testing of the two designs differed. The μ 370's objective was to place a fully functional System/370 on a single chip, while observing such technology constraints as number of wired circuits, power, and I/O pins. The D370's objective was to design a high-performance System/370 sensitive to technology constraints, but independent of power and number of I/O pins. The μ 370 was produced as a working chip, whereas the D370 is only a paper design.

6.3.2 The ALUs. The number, size, and type of functions supported by the ALUs in the two designs differed. The D370's design had one extra ALU that can be directly traced to the implementation of floating-point operations, while the μ 370 planned a separate floating-point chip. In addition, the D370 implemented the dynamic address translation hardware, while the μ 370 supplied a shifter that had a few complex shift patterns to aid in calculating the virtual address by using microcode.

The DAA does a high-level floor layout to help decide how to partition the algorithmic description, but this does not currently allow exclusion of functionality; the whole algorithmic description is implemented. Changes that modify the algorithmic description by including or excluding functionality are best made by changing the initial description or by having a postprocessor feed size constraints to the DAA.

The ALUs also differed in size. The μ 370 serialized the 32-bit and 64-bit operations of the System/370 architecture into four or eight cycles through an eight-bit ALU. The DAA's constraints were set to design a high-performance processor, and thus the data paths were not serialized. Less than a dozen rules could be added to the DAA to allow it to serialize on ALU width. However, this change would be better made by adding a transformation to the CMU/DA system that removes a single-abstract data flow operation, and replaces it with several smaller ones.

Finally, the ALUs differed in the functions they provided. The μ 370 has an ALU that can ADD and SUBTRACT packed-decimal numbers; the D370 performed these operations by adding hardware and microcode. The D370 has an ALU that can MULTIPLY, while the μ 370 MULTIPLIED by SHIFTING and ADDING. Davis felt the choice of an ALU that could MULTIPLY was reasonable and consistent with the constraints used by the DAA.

6.3.3 *The buses.* The designs differed in the number, size and type of buses used. The μ 370's eight-bit buses and 16-bit bus serve the same purpose as D370's eight-bit and 64-bit buses. The size difference is accounted for by the μ 370 serializing the 32-bit and 64-bit operations of the System/370 architecture down to eight-bit operations, as discussed above. Also, the D370 uses bidirectional buses, where the μ 370 used separate fan-in and fan-out buses. Davis felt the design choices made by the DAA were reasonable for the higher-performance D370 design, citing the IBM System/370 model 158 as an example of this style of busing.

6.3.4 *The memories.* The memory functional blocks differed only slightly. The D370 uses an eight-byte buffer with a four-byte memory data register, while the μ 370 uses eight bytes of memory data registers. Davis felt this and even more elaborate cache schemes suited the higher-performance processors. He suggested the D370 use dual-ported memories for its general-purpose registers, to allow the use of two registers during the same cycle. Dual-ported memories would require a few rule changes, but would allow up to two memory array accesses during the same clock cycle. The rules for finding the address, input and output ports of memories would have to be enhanced to check for idle ports. All told, about 20 rules would have to be modified or extended to effect this change.

6.3.5 *The registers.* Both descriptions have about the same number of bytes of architectural and temporary registers. However, the μ 370 groups all the architectural registers off chip in a fast local store, which can be thought of as memory. This would be a major change in the structure of the DAA. However, it could be accomplished simply, as a post-processor pass by the CMU/DA system as other technology-specific hardware is bound to the modules.

6.4 Summary of The System/370 Experiment

This chapter has shown the generality of design knowledge in the DAA by comparing and contrasting an IBM System/370 designed by an expert human designer, Claud Davis, against the design produced by the DAA. The differences were either explained and shown to be unimportant, or changes to the system were discussed. Davis felt the design produced by DAA exhibited the quality he would expect from one of his better designers.

This chapter has shown the first large implementation design, automatically generated from an algorithmic description and constraints, that has been favorably critiqued by an expert designer. Furthermore, the design required 47 hours of CPU time, which with some work can be reduced by a factor of 12 to about 4 hours of CPU time. This clearly shows the dramatic improvement in CPU time for large designs obtained using methods that replace backtracking by match techniques. Finally, because this design was generated using synthesis techniques, it is possible to verify its operation by construction⁹⁰ and link it to the rest of the CMU/DA design environment.

Chapter 7: CONCLUSION

This thesis has shown how expert VLSI designers choose a MOS microcomputer's implementation and how a knowledge-based expert-system, the DAA, can mimic their results. It begins by showing that the KBES technique is a promising approach to design synthesis. Using a KBES approach has sped the development of the DAA by providing a framework that allows incremental addition of common-sense modular design knowledge and queries about the knowledge during the design task. This framework has increased the flexibility of the final system and reduced the execution time by replacing backtracking techniques with match techniques (the D370 design time can be reduced to about 4 hours of CPU time). Like a human designer, the DAA has become a better designer as its rule memory expands. The extraction, codification and testing of the expert designers' knowledge has facilitated a better understanding of VLSI design-synthesis, while providing another KBES system for computer scientists and knowledge engineers to examine. This KBES approach has opened the door to a whole new class of intelligent computer aided design tools.

The thesis has also examined how human designers are constantly partitioning the design task into more manageable subtasks. These tasks involve global allocation of architectural modules and registers, local allocation of control steps, operators, registers data paths and control, global allocation of operators and registers, and global improvements to the design. The designers retain a global view of the hardware by using high-level floor plans concerned with connectivity and functionality, that are updated as the design proceeds. More important than minimizing components, designers strive for testable synchronous designs that will lay out well. Thus, the thesis has shown that creating a good design is not just minimizing components, but paying careful attention to testability and

connectivity.

The DAA has created efficient, testable and usable designs of many processors including the MCS6502 and the IBM System/370. The design of the MCS6502 was critiqued during the knowledge acquisition development phase of the DAA and thought to be a good design. A design of each of the small ISPS descriptions maintained at CMU showed that the system would produce a functionally correct design for a large number of test cases. The IBM System/370 was critiqued at IBM and exhibited the quality expected from one of IBM's better designers. This is the first research effort in implementation allocation that has been verified by expert designers as producing quality designs. We are no longer comparing designs by counting the number of components, but having experts comment on the quality of the design.

Finally, because the DAA is incorporated into the CMU/DA synthesis environment it is possible to verify a design's operation by construction, decrease the time it takes to design a chip, automatically provide multi-level documentation for the finished design, and create reliable and testable designs.

However, the research does not end here. The DAA system could be expanded and improved in a number of ways by adding more digital design knowledge or by improving the underlying expert system. New styles of allocation could be incorporated into the DAA such as pipeline, fault tolerant, or digital signal processing architectures. New transformations for minimizing bit widths could be added to the VTDRIVE and the DAA. New user interaction and constraint handling could be added to the DAA. New underlying expert systems could be used to increase the speed of designs (OPS83 is currently claimed to yield an increase in speed between 20 to one and 50 to one, which would open the door to

an interactive DAA environment). New faster rule antecedents ordering could be achieved by dynamically changing the discrimination net at run time or providing the necessary measurement tools for rule performance.

Looking to the far horizon, a knowledge-acquisition system limited to implementation design could be developed, which allows expert designers to add and modify design rules and test examples for the DAA system. This would allow the personalization of the rule base for each designer. Looking further, the knowledge in the DAA could be used to develop a teaching environment for beginning designers. Finally, the DAA could be extended to start with a design, critique it, and help a designer modify it to meet constraints.

REFERENCES

- [1] Mead, C. and Conway, L., *Introduction to VLSI systems*, Addison-Wesley Publishing Company, Reading, Massachusetts (1980).
- [2] Director, S. W., Parker, A. C., Siewiorek, D. P., and Thomas, D. E., "A design methodology and computer aids for digital VLSI systems," *IEEE Transactions on Circuits and Systems* cas-28(7) (July, 1981).
- [3] Thomas, D. E., Hitchcock, C. Y. III, Kowalski, T. J., Rajan, J. V., and Walker, R., "Automatic Data Path Synthesis," *Computer* 16(12) pp. 59-70 (December, 1983).
- [4] Marwedel, P. and Zimmermann, G., *MIMOLA Software System User Manual*, 1, Institut Fur Informatik und Praktische Mathematik, Christian-Albrechts-Universitat Kiel (May, 1979).
- [5] Hafer, L. J., *Automated data-memory synthesis: A formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic*, PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University (June, 1981). Also in Design Research Center DRC-02-05-81
- [6] Hafer, L., *Data - Memory Allocation in the Distributed Logic Design Style*, Masters thesis, Carnegie-Mellon University (December 21, 1977).
- [7] Hitchcock, C. Y. III, "Automated Synthesis of Data Paths," CMUCAD-83-4, SRC-CMU Center for Computer-Aided Design, Carnegie-Mellon University (January, 1983).
- [8] Tseng, C. J. and Siewiorek, D. P., "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proceedings of the Twentieth Design Automation Conference*, pp. 490-496 (June 27, 1983).
- [9] Feigenbaum, E. A., *Knowledge Engineering: The Applied Side of Artificial Intelligence*, Computer Science Department, Stanford University (1980).
- [10] Siewiorek, D. P., Bell, C. G., and Newell, A., *Computer Structures: Principles and Examples*, McGraw-Hill, New York (1982).
- [11] Brooks, F. P. Jr., *The Mythical Man-Month*, Addison-Wesley, Reading, Mass. (1975).
- [12] Thomas, D. E., "The Automatic Synthesis of Digital Systems," *Proceedings of the IEEE* 69(10) pp. 1200-1211 (October, 1981).
- [13] Hafer, L. J., "Automated Synthesis of Digital Hardware," *IEEE Transactions on Computers* C-31(1) pp. 93-109 (January, 1982).
- [14] Zimmermann, G., "The MIMOLA design system: a computer aided digital processor design method," *Proceedings of the Sixteenth Design Automation Conference*, pp. 65-72 ACM SIGDA and IEEE Computer Society DATC, (June 1979).
- [15] Barbacci, M. R., Nagle, A. W., and Northcutt, J. D., *An ISPS Simulator*, Department of Computer Science, Carnegie-Mellon University (January 4, 1980).
- [16] Barbacci, M. R., Barnes, G. E., Cattell, R. G., and Siewiorek, D. P., *The ISPS Computer Description Language*, Department of Computer Science, Carnegie-

Mellon University (August 16, 1979).

- [17] Snow, E. A., *Automation of Module Set Independent Register-Transfer Level Design*, PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University (April 1978).
- [18] McFarland, M. C., "The VT: A Database for Automated Digital Design," DRC-01-4-80, Design Research Center, Carnegie-Mellon University (December 1978).
- [19] Vasantharajan, J., *Design and Implementation of a VT-Based Multi-level Representation*, Masters thesis, Department of Electrical Engineering, Carnegie-Mellon University (February 10, 1982).
- [20] Bushnell, M., Geiger, D., Kim, J., LaPotin, D., Nasif, S., Nestor, J., Rajan, J., Surjwas, A., and Walker, H., "DIF: The CMU-DA Intermediate Form," CMUCAD-83-11, Department of Electrical and Computer Engineering, Carnegie-Mellon University (July, 1983).
- [21] Wulf, W. A., Johnson, R. K., Weinstock, C. B., Hobbs, S. O., and Geschke, C. M., *The Design of an Optimizing Compiler*, North Holland, New York (1977).
- [22] Thomas, D. E., *The Design and Analysis of an Automated Design Style Selector*, PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University (April 1977).
- [23] Sakallah, K. and Director, S. W., "An Activity-Directed Circuit Simulation Algorithm," *Proceedings of the 1980 IEEE International Conference on Circuits and Computers*, pp. 1032-1035 IEEE, (October, 1980).
- [24] Leive, G. W., *The Binding of Modules to Abstract Digital Hardware*, PhD Thesis Proposal, Department of Electrical Engineering, Carnegie-Mellon University
- [25] Leive, G. W. and Thomas, D. E., *Module DataBase - User's guide*, Second Edition, Department of Electrical Engineering, Carnegie-Mellon University (October 1980).
- [26] Thomas, D. E. and Leive, G. W., "Automating Technology Relative Logic Synthesis and Module Selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 94-105 IEEE, (April, 1983).
- [27] Maly, W., Strojwas, A. J., and Director, S. W., "Fabrication Based Statistical Design of Monolithic IC's," *Proceedings of the International Symposium on Circuits and Systems*, (April, 1981).
- [28] Forgy, C. L., *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University (July, 1981).
- [29] Gillogly, J. J., "Performance Analysis of the Technology Chess Program," CMU-CS-78-109, Department of Computer Science, Carnegie-Mellon University (March 1978).
- [30] Di Russo, R., *A Design Implementation Using the CMU-DA System*, Masters thesis, Department of Electrical Engineering, Carnegie-Mellon University (October 20, 1981).

- [31] Parker, A. C., Thomas, D. E., Siewiorek, D. P., Barbacci, M. R., Leive, G., and Kim, J., "The CMU Design Automation System: an example of automated data path design," *Proceedings of the Sixteenth Design Automation Conference*, pp. 73-80 ACM SIGDA and IEEE Computer Society DATC, (June 1979).
- [32] Aho, A. V., Johnson, S. C., and Ullman, J. D., "Code Generation for Expressions with Common Subexpressions," *Journal of the ACM* 21(1) p. January, 1977 (146-160). Also in *ACM Symposium on Principles of Programming Languages*, pp 19-31, 1976.
- [33] McFarland, M. C., *Allocating Registers, Processors and Connections*. (September 5, 1981).
- [34] Brown, H., Tong, C., and Foyster, G., "Palladio: An Exploratory Environment for Circuit Design," *Computer* 16(12) pp. 41-56 (December, 1983).
- [35] Newell, A. and Simon, H. A., "GPS, A Program that Simulates Human Thought," pp. 279-293 in *Computers and Thought*, ed. Feigenbaum, E. A. and Feldman, J. A. McGraw-Hill, New York (1963).
- [36] Duda, R. O. and Gaschnig, J. G., "Knowledge-Based Expert Systems Come of Age," *Byte* 6(9) pp. 238-281 (September, 1981).
- [37] Feigenbaum, E. A., "The art of artificial intelligence: I. Themes and case studies of knowledge engineering," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 1014-1029 Massachusetts Institute of Technology, (1977). Also in *Expert Systems in the Microelectronic Age*, Michie, D. (Ed.) Edinburgh: Edinburgh University Press, 1979, pp. 3-25. and in *AFIPS Conference Proceedings*, Vol. 47, pp. 227-240 (June, 1978).
- [38] Rychener, M. D., "Knowledge-Based Expert Systems: A Brief Bibliography," CMU-CS-81-127, Department of Computer Science, Carnegie-Mellon University (June 26, 1981).
- [39] McDermott, J. D., *Private Communication*. (March, 1984).
- [40] Davis, R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules," *Artificial Intelligence* 12 pp. 121-157 (August, 1979).
- [41] Clancey, W. J., "Tutoring rules for guiding a case method dialogue," *International Journal of Man-Machine Studies* 11 pp. 25-49 (1979). Also in *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (1979), pp. 155-161.
- [42] Pople, H. E., Jr. and et al., "DIALOG: A Model of Diagnostic Logic for Internal Medicine," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 848-855 (September, 1975).
- [43] Shortliffe, E. H., *Computer Based Medical Consultations: MYCIN*, Elsevier, New York (1976).
- [44] Clancey, W. J., Shortliffe, E. H., and Buchanan, B. G., "Intelligent Computer-Aided Instruction for Medical Diagnosis," *Proceedings of the 3rd Symposium on Computer Application in Medical Care*, pp. 175-183 (1979).

- [45] Fagan, L., Kunz, J., Feigenbaum, E. A., and Osborn, J., "Representation of Dynamic Clinical Knowledge: Measurement Interpretation in the Intensive Care Unit," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 260-262 (August 20-23, 1979).
- [46] Bennett, J. S. and Englemore, R. S., "SACON: A knowledge-based consultant for structural analysis," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 47-49 Tokyo, (August 20-23, 1979).
- [47] Duda, R. O., Gaschnig, J., and Hart, P. E., "Model design in the Prospector system for mineral exploration," pp. 153-167 in *Expert Systems in the Micro Electronic Age*, ed. Michie, D., Edinburgh University Press, Edinburgh (1979). Also in [Waterman-1978], pp. 203-222.
- [48] Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J., "On Generality and Problem Solving: A Case Study Using the DENDRAL Program," pp. 165-190 in *Machine Intelligence 6*, ed. Meltzer, B. and Michie, D., American Elsevier, New York (1971).
- [49] Wipke, W. T., "Computer Planning of Research in Organic Chemistry," pp. 381-391 in *Computers in Chemical Education and Research*, ed. Ludena, E. V. Sabelli, N. H. and Wahl, A. C., Plenum Press, New York (1976).
- [50] Gelernter, H. L. and et al., "Empirical Explorations of SYNCHEM," *Science*, pp. 1041-1049 (September 9, 1977).
- [51] Grinberg, M. R., "A knowledge based design system for digital electronics," *Proceedings of the First Annual National Conference on Artificial Intelligence*, pp. 283-285 AAAI, (1980).
- [52] Stallman, R. M. and Sussman, G. J., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence 9* pp. 135-196 (1977).
- [53] Brown, J., Barton, R., Bell, A. G., and Bobrow, R. J., "SOPHIE: A sophisticated instructional environment," AD-A010109 (December 1974). Distributed by NTIS (Dept. of Commerce).
- [54] Stefik, M. J., "Inferring DNA Structures from Segmentation Data," *Artificial Intelligence 11* pp. 85-114 North-Holland, (August, 1978).
- [55] deKloer, J., "Qualitative and Quantitative reasoning in Classical Mechanics," pp. 11-30 in *Artificial Intelligence: An MIT Perspective*, ed. Winston, P. H. and Brown, R. H., Massachusetts Institute of Technology, Cambridge, Massachusetts (1979).
- [56] Barrow, D. R., "An Experiment in Knowledge-Based Automatic Programming," *Artificial Intelligence 12* pp. 7-1119 (August, 1979).
- [57] Bennett, J. S. and Hollander, C. R., "Dart: An Expert System for Computer Fault Diagnosis," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 843-845 (1981).
- [58] McDermott, J., "RI: An Expert in the Computer Systems Domain," *Proceedings of the First Annual National Conference on Artificial Intelligence*, pp. 269-271

(1980).

- [59] McDermott, J., "Domain Knowledge and the Design Process," *Proceedings of the 18th Design Automation Conference*, pp. 580-588 ACM IEEE, (June 29, 1981).
- [60] Chisholm, I. H. and Sleeman, D. H., "An Aide for Theory Formulation," pp. 202-212 in *Expert Systems in the Micro Electronic Age*, ed. Michie, D., Edinburgh University Press, Scotland (1979).
- [61] Erman, L. D. and Lesser, V. R., "HEARSAY-II: Tutorial Introduction and Retrospective View," CMU-CS-78-117, Department of Computer Science, Carnegie-Mellon University (May 1978).
- [62] Nil, H. P. and Feigenbaum, E. A., "Rule-Based Understanding of Signals," pp. 483-501 in *Pattern-Directed Inference Systems*, ed. Waterman, D. A. and Hayes-Roth, F., Academic Press, New York (1978).
- [63] van Melle, W., "A Domain Independent Production-Rule System for Consultation Programs," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 923-925 (August 20-23, 1979).
- [64] Weiss, S. M. and Kulikowski, C. A., "EXPERT: A System for Developing Consultation Models," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 942-947 (August 20-23, 1979).
- [65] Balzer, R., Erman, L. D., London, P., and Williams, C., "HEARSAY-III: A Domain-Independent Framework for Expert Systems," *Proceedings of the First Annual National Conference on Artificial Intelligence*, pp. 108-110 (1980).
- [66] Reboh, R., "The Knowledge Acquisition System," Final Report, SRI Project 6415, Artificial Intelligence Center, SRI International, Menlo Park, CA (September, 1979). Duda, R. O., et al. "Also in 'A Computer-Based Consultant for Mineral Exploration,' Duda, R. O., et al.
- [67] Buchanan, B. G. and Feigenbaum, E. A., "DENDRAL and Meta-DENDRAL: their applications dimensions," *Artificial Intelligence* 11 pp. 5-24 (1978).
- [68] Clancey, W. J. and Letsinger, R., "Neomycin: Reconfiguring a Rule Based Expert System for Application to Teaching," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 829-835 (1981).
- [69] deKleer, J., Doyle, J., Steele, G. L., and Sussman, G. J., "Explicit Control of Reasoning," pp. 93-116 in *Artificial Intelligence: An MIT Perspective*, ed. Winston, P. H. and Brown, R. H., Massachusetts Institute of Technology, Cambridge, Massachusetts (1979).
- [70] Rychener, M. D., *OPS3 production system language tutorial and reference manual*, Department of Computer Science, Carnegie-Mellon University (1980).
- [71] Waterman, D. A., "User-Oriented Systems for Capturing Expertise: A Rule-Based Approach," pp. 26-34 in *Expert Systems in the Microelectronic Age*, ed. Michie, D., Edinburgh University Press, Edinburgh (1979).
- [72] Ditzel, D., *Private Communication*. (July, 1981).

- [73] Maul, M., *Private Communication*. (July, 1981).
- [74] Williams, G., *Private Communication*. (July, 1981).
- [75] Wilson, A., *Private Communication*. (July, 1981).
- [76] Bourne, S. R., *UNIX Circuit Design System*, First Edition, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey (December 1980).
- [77] Newell, A., "Heuristic programming: ill-structured problems," pp. 360-414 in *Progress in Operations Research 3*, ed. Aronofsky, J., Wiley, New York (1969).
- [78] Rose, M. A., *Structured Control Flow: An Architectural Technique for Improving Control Flow Performance*, Masters thesis, Department of Electrical Engineering, Carnegie-Mellon University (November, 1983).
- [79] Gatesby, D. A., *Digital Design from an Abstract Algorithmic Representation: Design and Implementation of A Framework for Interactive Design*, Masters thesis, Department of Electrical Engineering, Carnegie-Mellon University (October, 1981).
- [80] Kernighan, B. W. and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Sys. Tech. J.* 49(2) pp. 291-308 (1970).
- [81] Schweikert, D. G. and Kernighan, B. W., "A Proper Model for the Partitioning of Electrical Circuits," *Proc. 9th Design Automation Workshop*, pp. 57-62 ACM IEEE, (1972).
- [82] Breuer, M. A., "A Class of Min-Cut Placement Algorithms," *Proc. 13th Design Automation Workshop*, pp. 284-290 ACM IEEE, (1976).
- [83] Payne, T. S. and vanCleave, W. M., "Automated Partitioning of Hierarchically Specified Digital Systems," *Proc. 19th Design Automation Conf.*, pp. 182-192 ACM IEEE, (1982).
- [84] Sokol, R. S. and Sneath, P. H. A., *Principles of Numerical Taxonomy*, W. H. Freeman and Co., San Francisco (1963).
- [85] McFarland, M. C., "Computer-Aided Partitioning of Behavioral Hardware," *Proceedings of the Twentieth Design Automation Conference*, pp. 472-478 (June, 1983).
- [86] Kowalski, T. J., *Computer-Aided Cost Estimation from Implementation Specifications*. (unpublished).
- [87] Davis, C., Maley, G., Simmons, R., Stroller, H., Warren, R., and Wehr, T., "IBM System/370 Bipolar Gate Array Micro-Processor Chip," *Proceedings of the International Conference on Circuits and Computers 2(2)* pp. 669-673 (October, 1980).
- [88] Davis, C., *Personal letter to Dr. D. E. Thomas*. (August 12, 1983).
- [89] Kowalski, T. J., *The VLSI Design Automation Assistant: The IBM 370 Critique*, Department of Electrical and Computer Engineering, Carnegie-Mellon University (September, 1983).

- [90] McFarland, M. C., *Mathematical Models for Verification in a Design Automation System*, Phd thesis, Carnegie-Mellon University (July, 1981). Also in Design Research Center DRC-02-06-81

Appendix A: WORKING-MEMORY VT

The following working-memory listing shows the same decoding loop as the ISPS and VT descriptions in Figures 18 and 19. Parentheses are used to delimit the boundaries of working-memory elements. The type of the working-memory element is given by the first name after the opening parenthesis. It corresponds to the names following the keyword *literalize* in Tables 6 and 7. The type is followed by its attribute-value pairs. By looking at the ISPS and VT descriptions you can see the direct translation of the ISPS carriers CPAGE and I into VT-bodies, constants and expression results into outnodes, and the procedure EADD into a VT-body with a list of operators and outnodes. This is the form in which the design task is given to DAA.

```
(vbody "id s1" type s "isp-name EXAMPLE" "vt 1)
(vbody "id r2" type r "parent s1" "bit-left 0" "bit-right 4" "isp-name CPAGE" "vt 2)
(vbody "id r3" type r "parent s1" "bit-left 0" "bit-right 11" "isp-name I" "vt 3)
(outnode "id ".c1" type c "bits 4" "value 7" "vt 0)
(outnode "id ".c2" type c "bits 5" "value 0" "vt 0)
(outnode "id ".c3" type c "bits 1" "value 0" "vt 0)
(outnode "id ".c4" type c "bits 4" "value 0" "vt 0)
(vbody "id v6" type v "parent s1" "entity-flag global" "bit-left 0" "bit-right 11" "isp-name
  EADD" "inputs Iv6" "operators OPv6" "outputs Ov6" "calls Cv6" "opcalls OPCv6"
  "attributes ATv6" "control-steps-assigned t" "vt 6)
(lists "id Iv6" "list v6.i1 v6.i2 v6.i3" "vt 6)
(outnode "id v6.i1" type i "carrier-id r3" "bits 12" "isp-name I" "vt 6)
(outnode "id v6.i2" type i "carrier-id v6" "bits 12" "isp-name EADD" "vt 6)
(outnode "id v6.i3" type i "carrier-id r2" "bits 5" "isp-name CPAGE" "vt 6)
(lists "id OPv6" "list v6.x1 v6.x2 v6.x3" "vt 6)
(operator "id v6.x1" "opcode BIT-R" "inputs Iv6.x1" "outputs Ov6.x1" "control-step-begin 1"
  "control-step-end 1" "parent v6" "vt 6)
(lists "id Iv6.x1" "list v6.i1 ".c1" "vt 6)
(lists "id Ov6.x1" "list v6.x1.p1" "vt 6)
(outnode "id v6.x1.p1" type p "bits 1" "isp-name PB" "vt 6)
(operator "id v6.x2" "opcode SELECT" "inputs Iv6.x2" "outputs Ov6.x2" "control-step-begin
  1" "control-step-end 1" "parent v6" "branches Bv6.x2" "vt 6)
(lists "id Iv6.x2" "list v6.x1.p1" "vt 6)
(lists "id Ov6.x2" "list v6.x2.p1" "vt 6)
(outnode "id v6.x2.p1" type p "carrier-id v6" "bits 12" "isp-name EADD" "vt 6)
(lists "id Bv6.x2" "list v6.x2.b1 v6.x2.b2" "vt 6)
(branch "id v6.x2.b1" type b "parent v6.x2" "activations ACv6.x2.b1" "operators
  OPv6.x2.b1" "inputs Iv6.x2.b1" "vt 6)
(lists "id ACv6.x2.b1" "list 0:0" "vt 6)
(lists "id OPv6.x2.b1" "list v6.x3 v6.x4 v6.x5" "vt 6)
(operator "id v6.x3" "opcode BIT-R" "inputs Iv6.x3" "outputs Ov6.x3" "control-step-begin 2"
  "control-step-end 2" "parent v6" "vt 6)
(lists "id Iv6.x3" "list v6.i1 ".c4" "vt 6)
(lists "id Ov6.x3" "list v6.x3.p1" "vt 6)
(outnode "id v6.x3.p1" type p "bits 7" "isp-name PA" "vt 6)
(operator "id v6.x4" "opcode @" "inputs Iv6.x4" "outputs Ov6.x4" "control-step-begin 2
```

```

    *control-step-end 2 *parent v6 *vt 6)
(lists "id lv6.x4" "list *.c2 v6.x3.pl" *vt 6)
(lists "id Ov6.x4" "list v6.x4.pl" *vt 6)
(outnode "id v6.x4.pl" "type p" "bits 12" *vt 6)
(operator "id v6.x5" "opcode GBIT-W" "inputs lv6.x5" "outputs Ov6.x5" *control-step-begin
    2 *control-step-end 2 *parent v6 *vt 6)
(lists "id lv6.x5" "list v6.i2 v6.x4.pl" *.c3 *vt 6)
(lists "id Ov6.x5" "list v6.x5.pl" *vt 6)
(outnode "id v6.x5.pl" "type p" "carrier-id v6" "bits 12" "isp-name EADD" *vt 6)
(lists "id lv6.x2.b1" "list v6.x5.pl" *vt 6)
(branch "id v6.x2.b2" "type b" "parent v6.x2" "activations ACv6.x2.b2" "operators
    OPv6.x2.b2" "inputs lv6.x2.b2" *vt 6)
(lists "id ACv6.x2.b2" "list 1:1" *vt 6)
(lists "id OPv6.x2.b2" "list v6.x6 v6.x7 v6.x8" *vt 6)
(operator "id v6.x6" "opcode BIT-R" "inputs lv6.x6" "outputs Ov6.x6" *control-step-begin 3
    *control-step-end 3 *parent v6 *vt 6)
(lists "id lv6.x6" "list v6.i1" *.c4 *vt 6)
(lists "id Ov6.x6" "list v6.x6.pl" *vt 6)
(outnode "id v6.x6.pl" "type p" "bits 7" "isp-name PA" *vt 6)
(operator "id v6.x7" "opcode @" "inputs lv6.x7" "outputs Ov6.x7" *control-step-begin 3
    *control-step-end 3 *parent v6 *vt 6)
(lists "id lv6.x7" "list v6.i3 v6.x6.pl" *vt 6)
(lists "id Ov6.x7" "list v6.x7.pl" *vt 6)
(outnode "id v6.x7.pl" "type p" "bits 12" *vt 6)
(operator "id v6.x8" "opcode GBIT-W" "inputs lv6.x8" "outputs Ov6.x8" *control-step-begin
    3 *control-step-end 3 *parent v6 *vt 6)
(lists "id lv6.x8" "list v6.i2 v6.x7.pl" *.c3 *vt 6)
(lists "id Ov6.x8" "list v6.x8.pl" *vt 6)
(outnode "id v6.x8.pl" "type p" "carrier-id v6" "bits 12" "isp-name EADD" *vt 6)
(lists "id lv6.x2.b2" "list v6.x8.pl" *vt 6)
(operator "id v6.x9" "opcode LEAVE" "call v6" "inputs lv6.x9" *control-step-begin 3
    *control-step-end 3 *parent v6 *vt 6)
(lists "id lv6.x9" "list v6.x2.pl" *vt 6)
(lists "id Ov6" "list v6.o1" *vt 6)
(outnode "id v6.o1" "type o" "carrier-id v6" "bits 12" "isp-name EADD" *vt 6)
(lists "id Cv6" "list v6" *vt 6)
(lists "id OPCv6" "list v6.x9" *vt 6)
(lists "id ATv6" "list tc 12 !EFFECTIVE-ADDRESS!" *vt 6)

```

Appendix B: WORKING-MEMORY USER PARAMETERS

The following working-memory listing shows the default technology database and user parameters. If the user does not supply values for these parameters in a file named *daa.l*, these values are used. Because these values were suggested by designers of MOS microprocessors attempting to design fast processors, the use of hardware is unlimited. Parentheses are used to delimit the boundaries of working-memory elements. The type of the working-memory element is given by the first name after the opening parenthesis. It corresponds to the names following the keyword *literalize* in Table 8. The type is followed by its attribute-value pairs. By looking at Tables 9, 10, and 11 you can see the direct translation of the values into working-memory elements. This is the form in which the technology database and constraints are given to DAA.

```
(db-operator "hw-opcode PLUS "control-step-delay 1 "vt-opcode + "allocated 0
"maximum 999 "group ARITHMETIC)
(db-operator "hw-opcode MINUS "control-step-delay 1 "vt-opcode - "allocated 0
"maximum 999 "group ARITHMETIC)
(db-operator "hw-opcode MULT "control-step-delay 1 "vt-opcode * "allocated 0
"maximum 999 "group ARITHMETIC)
(db-operator "hw-opcode DIV "control-step-delay 1 "vt-opcode / "allocated 0 "maximum
999 "group ARITHMETIC)
(db-operator "hw-opcode MOD "control-step-delay 1 "vt-opcode MOD "allocated 0
"maximum 999 "group ARITHMETIC)
(db-operator "hw-opcode UMINUS "control-step-delay 1 "vt-opcode --- "allocated 0
"maximum 999 "group ARITHMETIC)
(db-operator "hw-opcode AND "control-step-delay 1 "vt-opcode AND "allocated 0
"maximum 999 "group LOGICAL)
(db-operator "hw-opcode EQV "control-step-delay 1 "vt-opcode EQV "allocated 0
"maximum 999 "group LOGICAL)
(db-operator "hw-opcode OR "control-step-delay 1 "vt-opcode OR "allocated 0
"maximum 999 "group LOGICAL)
(db-operator "hw-opcode XOR "control-step-delay 1 "vt-opcode XOR "allocated 0
"maximum 999 "group LOGICAL)
(db-operator "hw-opcode NOT "control-step-delay 1 "vt-opcode NOT "allocated 0
"maximum 999 "group LOGICAL)
(db-operator "hw-opcode EQL "control-step-delay 1 "vt-opcode EQL "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode NEQ "control-step-delay 1 "vt-opcode NEQ "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode LSS "control-step-delay 1 "vt-opcode LSS "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode LEQ "control-step-delay 1 "vt-opcode LEQ "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode GEQ "control-step-delay 1 "vt-opcode GEQ "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode GTR "control-step-delay 1 "vt-opcode GTR "allocated 0
"maximum 999 "group RELATIONAL)
(db-operator "hw-opcode TST "control-step-delay 1 "vt-opcode TST "allocated 0
```

```

*maximum 999 *group RELATIONAL)
(db-operator *hw-opcode SR0 *control-step-delay 1 *vi-opcode SR0 *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SR1 *control-step-delay 1 *vi-opcode SR1 *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SRD *control-step-delay 1 *vi-opcode SRD *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SRR *control-step-delay 1 *vi-opcode SRR *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SLO *control-step-delay 1 *vi-opcode SLO *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SLI *control-step-delay 1 *vi-opcode SLI *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SLD *control-step-delay 1 *vi-opcode SLD *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SLR *control-step-delay 1 *vi-opcode SLR *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SLI *control-step-delay 1 *vi-opcode SLI *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode SRI *control-step-delay 1 *vi-opcode SRI *allocated 0
*maximum 999 *group SHIFT)
(db-operator *hw-opcode 0 PAD *control-step-delay 0 *vi-opcode PAD0 *allocated 0
*maximum 999 *group WIRING)
(db-operator *hw-opcode SPAD *control-step-delay 0 *vi-opcode PADS *allocated 0
*maximum 999 *group WIRING)
(db-operator *hw-opcode FREAD *control-step-delay 0 *vi-opcode BIT-R *allocated 0
*maximum 999 *group WIRING)
(db-operator *hw-opcode FWRITE *control-step-delay 0 *vi-opcode BIT-W *allocated 0
*maximum 999 *group WIRING)
(db-operator *hw-opcode AREAD *control-step-delay -1 *vi-opcode WORD-R *allocated
0 *maximum 999 *group WIRING)
(db-operator *hw-opcode AWRITE *control-step-delay -1 *vi-opcode WORD-W
*allocated 0 *maximum 999 *group WIRING)
(db-operator *hw-opcode GFREAD *control-step-delay 0 *vi-opcode GBIT-R *allocated 0
*maximum 999 *group WIRING)
(db-operator *hw-opcode GFWRITE *control-step-delay 0 *vi-opcode GBIT-W *allocated
0 *maximum 999 *group WIRING)
(db-operator *hw-opcode GAREAD *control-step-delay -1 *vi-opcode GWORD-R
*allocated 0 *maximum 999 *group WIRING)
(db-operator *hw-opcode GAWRITE *control-step-delay -1 *vi-opcode GWORD-W
*allocated 0 *maximum 999 *group WIRING)
(db-operator *hw-opcode DELAY *control-step-delay 0 *vi-opcode DELAY *allocated 0
*maximum 999 *group SYNCHRONIZATION)
(db-operator *hw-opcode WAIT *control-step-delay 0 *vi-opcode WAIT *allocated 0
*maximum 999 *group SYNCHRONIZATION)
(db-operator *hw-opcode DWAIT *control-step-delay 0 *vi-opcode TIME AIT *allocated
0 *maximum 999 *group SYNCHRONIZATION)
(db-operator *hw-opcode STOP *control-step-delay 0 *vi-opcode STOP *allocated 0

```

```

    *maximum 999 *group SYNCHRONIZATION)
(db-operator *hw-opcode SELECT *control-step-delay -1 *vi-opcode SELECT *allocated 0
  *maximum 999 *group BRANCH)
(db-operator *hw-opcode ENDSEL *control-step-delay 0 *vi-opcode ENDSEL *allocated 0
  *maximum 999 *group BRANCH)
(db-operator *hw-opcode DIVERGE *control-step-delay 0 *vi-opcode DIVERGE
  *allocated 0 *maximum 999 *group BRANCH)
(db-operator *hw-opcode MERGE *control-step-delay 0 *vi-opcode MERGE *allocated 0
  *maximum 999 *group BRANCH)
(db-operator *hw-opcode ENTER *control-step-delay -1 *vi-opcode ENTER *allocated 0
  *maximum 999 *group CONTROL)
(db-operator *hw-opcode CALL *control-step-delay -1 *vi-opcode CALL *allocated 0
  *maximum 999 *group CONTROL)
(db-operator *hw-opcode PSTART *control-step-delay -1 *vi-opcode PSTART *allocated 0
  *maximum 999 *group CONTROL)
(db-operator *hw-opcode LEAVE *control-step-delay -1 *vi-opcode LEAVE *allocated 0
  *maximum 999 *group CONTROL)
(db-operator *hw-opcode RESTART *control-step-delay -1 *vi-opcode RESTART
  *allocated 0 *maximum 999 *group CONTROL)
(db-operator *hw-opcode RESUME *control-step-delay -1 *vi-opcode RESUME
  *allocated 0 *maximum 999 *group CONTROL)
(db-operator *hw-opcode TERMIN *control-step-delay -1 *vi-opcode TERMIN
  *allocated 0 *maximum 999 *group CONTROL)
(db-operator *hw-opcode NOOP *control-step-delay 0 *vi-opcode NOOP *allocated 0
  *maximum 999 *group NOOP)
(db-operator *hw-opcode UNDEF *control-step-delay 0 *vi-opcode UNDEF *allocated 0
  *maximum 999 *group UNDEFINED)
(db-operator *hw-opcode UNPREDICTABLE *control-step-delay 0 *vi-opcode
  UNPREDICTABLE *allocated 0 *maximum 999 *group UNPREDICTABLE)
(db-operator *hw-opcode PARITY *control-step-delay 1 *vi-opcode PARITY *allocated 0
  *maximum 999 *group PARITY)
(db-operator *hw-opcode IS.RUNNING *control-step-delay 0 *vi-opcode IS.RUNNING
  *allocated 0 *maximum 999 *group IS.RUNNING)
(db-operator *hw-opcode INC *control-step-delay 1 *vi-opcode INC *allocated 0
  *maximum 999 *group INC)
(db-operator *hw-opcode DEC *control-step-delay 1 *vi-opcode DEC *allocated 0
  *maximum 999 *group DEC)
(db-operator *hw-opcode CLEAR *control-step-delay 1 *vi-opcode CLEAR *allocated 0
  *maximum 999 *group CLEAR)
(db-operator *hw-opcode MUX *control-step-delay 0 *vi-opcode MUX *allocated 0
  *maximum 999 *group MUX)
(db-operator *hw-opcode DEMUX *control-step-delay 0 *vi-opcode DEMUX *allocated 0
  *maximum 999 *group DEMUX)
(db-operator *hw-opcode BUS *control-step-delay 0 *vi-opcode BUS *allocated 0
  *maximum 999 *group BUS)
(db-operator *hw-opcode ALU *control-step-delay 1 *vi-opcode ALU *allocated 0
  *maximum 999 *group ALU)
(db-operator *hw-opcode CONCAT *control-step-delay 0 *vi-opcode @ *allocated 0

```


*maximum 999 *group WIRING)
(fold *type ARITHMETIC *prox 0.5 *cost 0.5)
(fold *type LOGICAL *prox 0.5 *cost 0.75)
(fold *type REGISTER *prox 0.5 *cost 0.5)
(max-delay-per-control-step *delay 50)

Appendix C: WORKING-MEMORY SCS

The following working-memory listing shows the decode loop from the ISPS example in Figure 18 translated into SCS as in Figures 20 and 21. Parentheses are used to delimit the boundaries of working-memory elements. The type of the working-memory element is given by the first name after the opening parenthesis. It corresponds to the names following the keyword *literalize* in Tables 12 and 13. The type is followed by its attribute-value pairs. By looking at the ISPS and VT descriptions you can see the direct translation of the ISPS carriers CPAGE and I into modules, ports, and links. This is the form DAA uses for the designed architecture.

```
(module "id *.c2.constant" type CONSTANT "bit-left 4" "bit-right 0" "value 0" "opt-flag
global)
(module "id controller0" type CONTROLLER "bit-left 0" "bit-right 0" "opt-flag arch)
(module "id r2.register" type REGISTER "bit-left 0" "bit-right 4" "opt-flag bound)
(module "id r3.register" type REGISTER "bit-left 0" "bit-right 11" "opt-flag bound)
(module "id v6.input.mux" type OPERATOR "bit-left 11" "bit-right 0" "value MUX" "opt-
flag global)
(module "id v6.register" type REGISTER "bit-left 0" "bit-right 11" "opt-flag global)
(module "id v6.x4" type OPERATOR "bit-left 11" "bit-right 0" "value @ " "opt-flag bound)
(module "id v6.x7" type OPERATOR "bit-left 11" "bit-right 0" "value @ " "opt-flag bound)
(mux-port-count "module v6.input.mux" "icat 2)
(port "id *.c2.output" type OUTPUT "bit-left 4" "bit-right 0" "module *.c2.constant)
(port "id controller0.source0" type INPUT "bit-left 0" "bit-right 0" "module controller0)
(port "id r2.output" type OUTPUT "bit-left 4" "bit-right 0" "module r2.register)
(port "id r3.output" type OUTPUT "bit-left 11" "bit-right 0" "module r3.register)
(port "id v6.input" type INPUT "bit-left 11" "bit-right 0" "module v6.register)
(port "id v6.input.output" type OUTPUT "bit-left 11" "bit-right 0" "module v6.input.mux)
(port "id v6.input.source0" type INPUT "bit-left 11" "bit-right 0" "module v6.input.mux)
(port "id v6.input.source1" type INPUT "bit-left 11" "bit-right 0" "module v6.input.mux)
(port "id v6.x4.output" type OUTPUT "bit-left 11" "bit-right 0" "module v6.x4)
(port "id v6.x4.source0" number 0 type INPUT "bit-left 4" "bit-right 0" "module v6.x4)
(port "id v6.x4.source1" number 1 type INPUT "bit-left 6" "bit-right 0" "module v6.x4)
(port "id v6.x7.output" type OUTPUT "bit-left 11" "bit-right 0" "module v6.x7)
(port "id v6.x7.source0" number 0 type INPUT "bit-left 4" "bit-right 0" "module v6.x7)
(port "id v6.x7.source1" number 1 type INPUT "bit-left 6" "bit-right 0" "module v6.x7)
(controller-port-count "module controller0" "kcnt 1)
(link "source-port *.c2.output" "source-bit-left 4" "source-bit-right 0" "dest-port
v6.x4.source0" "dest-bit-left 4" "dest-bit-right 0)
(link "source-port r2.output" "source-bit-left 4" "source-bit-right 0" "dest-port v6.x7.source0
"dest-bit-left 4" "dest-bit-right 0)
(link "source-port r3.output" "source-bit-left 6" "source-bit-right 0" "dest-port v6.x4.source1
"dest-bit-left 6" "dest-bit-right 0)
(link "source-port r3.output" "source-bit-left 6" "source-bit-right 0" "dest-port v6.x7.source1
"dest-bit-left 6" "dest-bit-right 0)
(link "source-port r3.output" "source-bit-left 7" "source-bit-right 7" "dest-port
controller0.source0" "dest-bit-left 0" "dest-bit-right 0)
(link "source-port v6.input.output" "source-bit-left 11" "source-bit-right 0" "dest-port
```

131

```
v6.input "dest-bit-left 11" "dest-bit-right 0")
(link "source-port v6.x4.output "source-bit-left 11" "source-bit-right 0" "dest-port
v6.input.source1 "dest-bit-left 11" "dest-bit-right 0")
(link "source-port v6.x7.output "source-bit-left 11" "source-bit-right 0" "dest-port
v6.input.source0 "dest-bit-left 11" "dest-bit-right 0")
```

Appendix D: THE SYSTEM/370 CRITIQUE

This appendix lists relevant sections of the IBM System/370 critique by Claud Davis. Also present were Ralph Bahnon, Ted Kowalski, Sumit Dasgupta, and an unknown IBM employee. The font convention followed in this appendix displays Davis's remarks in *italic*, Bahnon's, Kowalski's, Dasgupta's and the unknown IBM employee's remarks in roman with the speaker's name in bold. If the speaker's name is omitted, the speaker is either Davis or Kowalski, depending on whether the font is *italic* or roman.

D.1 The Critique

Ted: "OK, let's look at the design then and see how this turned out. I think it would be best if everyone gets on one side of the table. It's hard to read it from both sides at once."

Claud: "Well, it doesn't matter to me if you put it upside down. I used to teach ..."

"All right."

"I can still read it either way."

"Well, let's see if I ..."

"I haven't really thought too much about the how I do design."

"Well, that's OK."

"I guess we just do it."

"I think most people just sort of do it. Let me explain what my drafting is really all about here. These boxes are associated with ... (perhaps I should do something about ...)"

"All right."

"Do you want some tape?"

Ralph: "You may tape it down after a while."

Ted: "Let's mark the orientation."

"OK, might as well orient the thing."

"It'll help later. I'll try to figure out how to put this thing back together again later. These are registers of different bit widths. That's a 64 bit register. Zero is the low order, and 63 is the high word."

"Yes."

"This is backwards. IBM does things backwards. I first learned to do a large operating system, on an IBM system. And then after six years of using DEC equipment, I finally stopped using hex and started using octal, and now the DEC community has changed it to the VAX and is back to hex, at least. The byte order is still different. The order is really screwed. It's like ah-b-h! Just take that to mean a 64 bit register. It doesn't matter which order the bits are in. So we have a number of registers. These registers here aren't connected because they're part of the channel controller. The design that I have here has floating point, it has all of the integer, packed and decimal instructions and such, and I omitted channel controller stuff because I thought that was a connection to the outside world. I don't know. In your design did you use floating point on your chip?"

Clark: "Yes. That was mostly handled in ROS. That's the reason for 17 working registers."

Te: "OK."

"Our problem in trying to evaluate that as a true 370 is going to be a little awkward because you have to bear in mind I was doing that to also run another architecture, namely, UCI which is a very simple machine except it's heavily register oriented, but we did those in a stack which is not shown, obviously, and so that data flow will handle either a 370. If it can handle a 370, it can handle sufficient complexity to handle any architecture I've run into so far. I've only done about three and since, I've dropped it. I haven't been working with it any more for about a year or so ago in a different area. But you will find that our busing and our registers are, in fact, more general purpose than you possibly would do. If you were really optimizing in 370 only. You implemented directly most of our architected registers, I presume?"

"Right."

"You got your gp's and you got your controls?"

"Right. The control registers are there; the general purpose register stack is there."

"Yes."

"So I have a whole bunch of those. My floating point registers ..."

"OK, you've got your fp's."

"Right. Let me get on the same side as you."

"Yes, get on the side with me and let's see where we are."

"I have a hard time reading upside down."

"I'm sorry for my designer who's Jerry, and he is apparently lost as usual."

"OK, so we have the storage keys ..."

"Yes, and we had those. We had ours in a stack. I can tell you where most of the stuff was."

"Why don't we write here that was in the stack."

"Yes, that was in the stack and that's due to space. Tell you why."

"All right. That's good."

"Normally I would not. Normally I would have implemented those."

"OK, let me tell you about the constraints that I used because designs change about constraints."

"What are we looking at here, 67 or is it 68?"

"Right, 68 goes to here. Constraints I used. I said, 'OK, I want this to be a fast design, so if there are times in which you can have multiple ALU's and increase the speed, do it. So you see, I actually have three ALU's. I also said there are some registers which I think should be architecturally defined, and those registers were like the DAT box."

Clark: "You may have noticed in our implementation, we couldn't implement the DAT box on the chip. Due to space, we have compare registers, and went back off and handle that in microcode and, again, that was due to space. We had a design with that, and it had to come out because of we couldn't wire the chip, well. No, we'd gone above my arbitrarily 5000 circuits. I laid down the law so that nobody goes above 5000 circuits."

Tok: "Why did you choose 5000 circuits?"

"Partly because of heat constraints."

"OK, so you had power constraint?"

"Yes, power constraints. We were in BIPOLAR."

"OK."

"And secondly, it's sort of like playing baseball where you only get one strike: you have to have hits."

"OK."

"We have the wire. There was no question, I had to wire that chip as far as I'm concerned, and I believe that I could wire 5000 circuits. And I didn't believe I could wire 7000, and we couldn't."

"OK, because your tools would blow up or ..."

"Well, no. Just not enough space. More space on that chip went to wires than went to circuits, so wire turned out to be the problem — not circuits."

"OK."

"Let's see. There were over 10,000 wires. So, it turned out to be a wire problem. So that's why in implementing our data we did the compare and as long as we stayed in the page, you'd never have to do anything more. Compares you can work with as long as you were in that page. And so we were working in addresses like that, and when you crossed the page boundaries, you'd have to do a translation and update. And that gave fairly decent performance, by the way. And those are the kind of tradeoffs in deciding what do you throw out and we threw that out and kept working registers aboard."

"OK, so we have the registers there now. I also have a primary memory."

"All right."

"And then we have this register and a buffer register — we actually have a double buffer register. I can get memory fetch ahead that way."

"OK, so you in essence cache 1 deep. We did not put a cache on ours due to cost, and that was all. It wasn't questioned. But, you see, you're building — let's see how wide your adders are. See, you got more silicon to work with. So you're building a larger machine. If I went to a larger machine, I would use a cache, a small machine like I was working with — you see, in the range we're in, if you're working with a machine in the 8 bit to 16 bit, they're cost oriented. Because you can get a good bit of performance. And as you go into your larger machines, then you become performance as your criteria and you shift, and your machine here looks like you're in performance rather than having cost predominate, you have to be cost sensitive, but it doesn't predominate, so in this design you

always have cache until memory cost pricing goes on for another five or six years, and then we may even throw it out."

Ted: "With constraints like that, the design takes too many hours to do multiple designs, when I fed in the constraints I said, well, I had a number of choices to make. Indeed, I ..."

Claud: "Consciously made performance."

"Consciously made performance, yes."

"All right."

"My predominant bit width through this whole thing is 3 busses: there's a 24 bit bus (which gets involved with the memory address register, and whenever it needs an address register, it seems to fall on the 24 bit bus). Most of my transfers are in 64 bits. I ended up needing the 64 bit bus, then all the 32 bit operations just fell on the bus and well, I'm just going to use only half the bus. I don't bring the wires over to the other half. And also have an 8 bit bus on which bytes are brought back and forth. What kind of busses did you have on your design?"

"All right. We had an 8 bit adder on it, and that's simply because it could keep up. I was memory bound, and that was 2 bytes wide."

"So that was 16 bits."

"Yes, that was 16 and we did have a 24 bit bus and a separate arithmetic unit for addresses. By the way, you'll discover that you do more arithmetic work calculating addresses than you do anything else."

"That's right."

"So we optimize the address calculation more than we did the other, simply to give us performance. So we had that and, let's see, what else? Predominantly we were, as I said we had a 16, 8, and 24 busses on that chip."

"A lot of my registers here are dealing with floating point, and you dealt with that on microcode in some sort."

"Yes, we did."

"So those things would all have gone to microcode?"

"Let me tell you, we did not optimize floating point at all. We were going to do a separate chip for floating point and did not simply because we didn't seem to find favor in the commercial side of the world with this company and so we were in the wrong division of the company. You see, there were people who had responsibility for designing computers in that performance range, and they were physically not located near where I was. So, anyway, we almost ignored floating point, so I'll put down low priority and planned a separate chip."

"So we cross off registers that were floating point."

"Now, in the speed range you're in, the chances are you would have floating point, and judging from what I'm seeing, you possibly would not optimize your floating point at the expense of data handling. Machines in this range normally are data-handling bound in performance. You'll discover that if you study even in the heavily optimized Fortran work,

that you can never get that percentage of work above 30. You just can't do it."

Ted: "So your feeling then is that these registers then are justified in a more Cadillac version performance machine."

Clark: "Yes. You see, when we started out, our objective was to put a 370 data flow on one chip. It's a bit of a puzzle if you can do it, and it was just a test to find out if we could do it. So, obviously, we were constrained by the amount of hardware we had and we decided to spend it in general, optimize the arithmetic operation of address generation, and then spend the rest in trying to hold data on chip so that you don't have to clutter up the memory bus which is the bottom line on this thing. And we spent our pins primarily in getting a parallel interface on to that chip so that we could handle all of our registers, in essence as parallel as needed for control and, secondly, if we had the pins, we didn't have to decode. We saved levels which gave us performance. That's the only way we could hold 100 nanoseconds. We could access our control store and bring it back and stay within the 100 nanoseconds."

"In your read only control store, what did you have out there, and also what was it controlled for?"

"For all registers. And the adder. It controlled the data flow and function. The functions, of course, being adders, shifters, and comparators. So it controlled not only function but the gates in order to do it and put away. We did not have multiple aperture registers (as I call them) where you got a stack where you can store two things in or read two at once. Now they're needed and they would possibly be needed in this for your gpr's. Oftentimes you need to read two gpr's at once, and if they're implemented in a stack --"

"Right now I have them as a single port memory of sorts."

"If you've got them individualized, why then fine. But if these are all implemented in what we'll call memory technology, and you can only read one of the registers per cycle, you're going to pay a penalty because normally you'll want to put a pair of those together. So if you'll look at the way 370 is done it pulls two registers and store one so that sort of thing is very useful when you can define it and you can very easily define it that way; the question is to hand it to a technologist and tell him to build you a few."

"You brought out an interesting point about when you deal with registers, you often grab two and store one."

"In the same cycle, right."

"In these ALU's what I have often is the registers come out and go to a bus. The bus has the ALU on it. It also has a register on it -- this register here. And it ended up being a general bus-stored register so the first phase of the bus would load this for one value then the other would load to another. Then they would do the ALU thing. You would actually bring dual ported registers out to your ALU's then. Is that what you're saying?"

D.2 The MCC Design

"That's the way we handle it. Let me show you (I don't have my design with me. I should remember it by now. I've worked on it long enough.)"

"Could we draw it on the plastic here?"

Claud: "I suppose we can, but it's been a while."

Ted: "I can't take the blackboard with me. The plastic would be best."

Samit: "Are you sure PEOPLExpress wouldn't mind the weight of the knowledge?"

"Off hand I'll try to lay it out a little bit. Now we had an arithmetic unit over here."

Ted: "This pen was writing fairly well for me. Would you try this one?"

"I'd as soon have one as the other. And, with that and now we had a decimal corrector immediately following it (decimal corrector/ladder). Now, we come down and we had what I would call a z-bus (it goes across). There were 17 registers through here. (I'll just start putting in the registers)."

"Now, this bus was capable of going into all of those. Of course, all of these are gated and it goes into (and I'll put down 17 over here for you). Now up here, I have two busses (come across, one here and then a second one there). All of these could go to one of them. It would read 'all.' The other would read 'selected ones.' This one could read 'selected registers.'"

"How'd you choose these two busses. Did you just decided you wanted ..."

"You're always trying to get a reasonable speed for the amount of investment in hardware. All right. Since we can put a good bit of our data here in these registers that we're working with, and if this is only 1 byte wide, then we have to ..."

"How wide are the registers?"

"These are just a byte. They're eight bits, and in handling the architecture of 32 bits (it's predominant in 370); therefore, you're going to make four cycles. OK, now. In doing that, we want to be sure we can finish 32 bits in four cycles, and in so doing, that says that you will need to feed both busses each cycle. So to do that, then it says you've got to have two independent busses, you've got to be able to independently gate from two registers: one to this bus and the other one. So you have to be reasonably careful about where you put things."

"Right, right."

"And we had a set of rules about where we put things that guarantee."

"What are the rules like?"

"We held down below. We would hold out ... We could group these up to three. Registers could be grouped into three and read out on to a 24 bit bus for address generation, and that bus and an adder were 24 bits, and you could read three and normally (I believe this was an increment of some sort), an increment of + or - 0, 1, 2, and 3."

"Is it easier for you to design the incrementer by any constant or small constant?"

"Small constants are much easier."

"And what would you define as a small constant - just those 0, 1, 2, 3 ... OK."

"The next one would go to 8, obviously."

138

Ted: "OK."

Clark: "It's not that difficult to go on up. It's just that we couldn't find much use for it."

"And if you want to go to 8, when would you think about going to 8? If you needed it, or if there was some cost involved, or is there some sort of tradeoff to make there?"

"Yes. We would ..."

"How much more expensive is an adder if you have to do up to 8 vs. only up to 3?"

"Not very much, really. It turned out we didn't find any use for it. That's the reason we didn't do it. It's not that much more expensive. It's actually bringing in another line. Your controls are possibly more expensive than this hardware."

"OK."

"And here in three registers, we would hold the address in memory, we hold the address in our stack which was off chip, held our gpr's, well, all of our architectural defines -- floating point registers and control registers ..."

"They're all off chip, OK."

"Control registers -- all in a very fast stack. I could still access it. Logically, it was on chip, because I could access it in a cycle -- the same with my controls. And so logically my controls down here lost ROS (read only store). It was also one cycle."

"Those were all parallel -- those on two chips?"

"This is 16 bits here. It's all I could do. 16 bits and I had enough buffer in here so where I could keep going. Down here we had -- what is it -- 51 lines plus parity plus 3 parity on my controls. And that says I can do all controls in parallel. I can do three things in parallel. I had some of these registers designated as IO (there was an IO bus that came in also -- this was my IO bus; it was 8 bits) and you could read into several different registers, I don't know how many."

"Excuse me. Your IO is also off chip. Over here in this design, by choosing not to input the channel stuff, that actually you had off chip anyway. How did you implement your channel control your start IO instruction?"

"That was all. This is the input bus. Actually there's another one of these chips. You can't build a controller for less than a chip."

"Sure, sure."

"It actually was another chip like this one. I changed the ROS."

"You had to interface here your channel controller with your address registers and those things, and those were all off-chip."

"Yes."

"And you access that off-chip through with this 8 bit IO?"

"Yes. So our IO's come in, so I could handle three things at once: you could do the arithmetic, you could also your address, and you could be doing IO."

139

Test: "You said address was the most predominant thing that you did. What sort of percentages would you assign to those things as to how much you were doing of each?"

Clawd: "Oh, gee, I can't give you percentages. I can say that's one, that's two, that's three. [Address, arithmetic & IO, respectively]"

"Great. Fine. OK."

"I really don't know, but that's the priority right off ..."

"Sure."

"... with no problem. Now, as far as the toughest thing in holding the 100 nanosecond cycle turned out to be this path."

"OK, from the ROS was the toughest."

"Not only do I generate the address for main memory, I generate the address for this one."

"Oh! You did both addresses! OK, why did you choose to do that?"

"Well, actually it's very seldom that you have to do ... excuse me. I didn't do it through here, I'm sorry. I did that with logic on chip because your address for your next order ROS word is normally sequential or it is included in a field."

"Ah, OK."

"Then you carry it there."

"Right."

"We carry it in a field within the word; therefore, the ROS was pretty self-contained in that respect. But we could change the word based on condition of the adder. If you had an overflow, underflow, our result was zero, and this was automatically set."

"OK."

"I also had a register here which was a condition register and I had eight lines that came out that could be set from some other like my IO controller, and that was a condition, and then I had eight (I guess there were eight more or actually four) there were four that could be set from my adder."

"You said four?"

"Yes -- that conditions, and that would be equals zero and overflow and I guess you'd call it underflow."

"Now, in my design I had actually a PSW word. If you took your PSW word, you spread out to the 8 bits here and then ..."

"I held part of my PSW here and the rest of it was up here. I look at this as the architecturally define."

"Right."

"Machine was here."

Test: "Most all the architecturally defined. OK."

Claw: "That's right. That was a stack. So we did that because this was a relatively small piece. Now, this was eight bits, obviously and these, of course, are eight. There are eight bits at this point back up here. This is 24 and this, of course, comes down and is 24 and goes back in to registers. Now, so you can begin to see that wires gave me problems -- as much difficulty as anything else."

"Sure. And so the reason why you kept the eight bit data path instead of having a 32 bit data path was just cost."

"We designed that at 16 and 32 and looked at it and we couldn't get any difference in performance."

"Oh! OK!"

"There was no performance difference because we had relatively fast circuits. They were in the order of 4 nanoseconds, and so we even allowed our logic level depth to limit up. Normally, I was going to ask you if you could run your logic levels. Our logic levels per cycle -- Normally in logic you will usually hold that down around 14. We were up at 24 and simply because we could do 24."

"OK, I don't know what the term 'logic levels per cycle' really means. Is that the number of transitions you make per clock phase or --"

"That's logic blocks."

"Oh, logic blocks."

"You know, like that and so on."

"OK."

"OK, I could do 24 of those in a 4 nanosecond cycle and so we did. Actually it was 3.6 but that would only be 96 nanoseconds and I was running at a 100 nanosecond cycle and the statistical distribution that's worse case for 96 -- statistical distribution. Since that's worse case, you're way back over there. It actually runs about 70."

"OK, I don't have a metric for logic level per cycle. You see, each one of these modules I have defined as to how much delay each block takes."

"That's good enough."

"Then I go and I add the number of blocks I had per clock -- AND'ing and OR'ing that stuff would all just go blah, blah, blah in one clock phase."

"OK."

"And it keeps track of how much it needs before it has to go to the next clock."

"As long as you know the delay through a function, that's sufficient because that's how I use the number level just to generate delay and that's all."

"Right. In fact one of the things that gave me a lot of grief actually in doing these rules is designers kept on saying, 'Every time you're done with a clock, you'd better have everything in registers or else I'm going to scream at you because I can't use LSSD techniques, I can't use testing techniques.' So some of my registers are there because I had to go and latch

values up to clock phase."

Claud: "By the way, I couldn't use LSSD on this because I didn't have enough blocks, and so what I did was I set up where I had a test port (a test port I call it), and that was so that I could put data in and set any register on that chip by my test port. And I would set them as a byte at a time, and I'd actually use one of the ports, but I had to have some micro code in order to do that. And I could go into test mode, and I could load any register broadside and then read it out. Therefore, I had no logic on here that I couldn't get to with the exception of implementing some of the console function that's architecturally defined, some pushbutton stuff. Because I don't know how you test -- When you push button and get only one pulse out."

Ted: "That's right."

"It's a real dog! In fact, I don't know how you can do that with DC tests."

"I don't know either."

"That was a piece of logic that we couldn't check."

"Sure."

"The rest of it, why we checked it that way. We used a test port rather than LSSD because we couldn't get in but we used the same patterns."

"Do you remember how many words you had in ROS?"

"Yes, I think I do. I believe it was 2000."

"2000?"

"Yes. 2K words."

"OK."

"It's two or three because it comes in 2K chunks. That's the reason I know. If it had come in three, we would have used three like on the Mod 50 which was my machine -- Model 50, 360 machine, while we used 2800 (2.8K) words which were a little wider than that -- words of ROS. Now if you look at that and say it's a waste of pins to go 51 wide on your control (and it would be if you were designing specifically for 370) and you can narrow that down. I didn't know what other architectures I was working with. This was an experimental chip, so you hold your options open. This gives me more variability, and so I did it that way. The reason it would be in real life you'd squeeze that down. Pins cost money one dollar per chip. They're expensive so -- A buck changes, of course. What I'm trying to say is --"

"I understand."

"They add significantly to the cost of your chip and so your next thing you mount your chip on, and then it goes on and on and that's why you'll notice we're using more pins than anybody else. But the reason they don't use them may not be that they don't know how to put more pins on the chip."

"The cost."

Clark: "It's that cost squeeze. They want to keep the cost down."

Test: "Actually I've screwed around with using my system to generate some other designs and, you know, one of the things my designers was telling me is that a system like this is wonderful because most times he'll find he's got area to spare on his chip because it's pin out limited. He can only put so much function on his chip because of his pins."

"OK."

"And so he doesn't care if he wastes a little area here or there as long as he can get fast turnaround in getting that architecture from the algorithm."

"Well, I was also pushing the number of pins. I was using more pins than we do on our commercial chip."

"How many pins did you use on the ICCO chip?"

"200."

"200 pins. OK."

"My original design was 240. They got nervous, so I cut it back to 200."

"OK."

"We can make it 240 pin chip."

"Something I've been working most recently on is terms of my rules in choosing busses: how do you choose how to put things on busses and not put things on busses."

"OK."

"How do you choose major bus widths? How do you do that?"

"That's due to the exclusivity of your data, like here. I decided I needed two inputs so I have to have two independent busses. Now the way that I would tell my compiler to handle temporary data is that you've got to look ahead and see what you're going to be doing in the next cycle and be sure that they can feed both busses. And that's the way I set up the gating."

"OK."

"And it was that set of rules that guided us in coding our ROS to decide, OK, where are you going to put your temporary data."

D.2.J Tape change.

"We started out, I had these constraints: 3000 logic blocks we had, I'll say 200 pins, and I had three watts. All right? That was what I had."

"Now how did you, OK, can you tell me how you went from watts and pins to blocks?"

"OK."

"How do they relate?"

"All right. Blocks come in two flavors: there's fast blocks that are hot."

Ted: "Sure."

Claud: "And then there's the slow ones."

"Sure."

"Slower. And when you need a fast one, or the drivers are hot. The drivers that drive the 200 pins, represented almost half the power of my chip."

"OK, sure."

"OK, so that's the first thing you look at. You say, well out of the 200 pins, how many are drivers? Rule of thumb is 40%."

Samit: "Then that's two to one then?"

"Not quite."

Samit: "Two out of five. Right?"

"Yeah. Not quite. A conservative guess is about 40%. And so when you first start out, you know, with a blank sheet of paper, that these are the kind of rules of thumb that you start with. And you start there and you say, well, oh, roughly 40% will be drivers. Some of them you'll have both drivers and receivers on. We had that capability of handling both drivers and receivers. Yeah, we could only have, I don't remember how many drivers we could have; it seemed to me like 140, something like that. Now that is open to question of my memory, but we had a lot of receivers. Anyway ..."

Samit: "Wasn't there, excuse me. Wasn't there also a limitation on the number of simultaneous switching drivers?"

"Yes."

Samit: "Yes. And 40% still put you way below that limit, right?"

"Yeah. No, no, not below the problem. No."

Samit: "It would still be way over that limit."

"Yeah. But never the less, you start out, you subtract that power and decide what you've got left for logic."

Ted: "OK."

"And then you immediately determine, all right, I can use 10% or 15% of the high power blocks."

"OK."

"Whatever percentage that is, and I don't remember what ours was, but I imagine that it would be in the 15% to 20% range."

"And then you use those type of things, you use those for buses?"

"You'll use those in driving this bus ..."

"OK."

Chuck: "... that bus or that bus."

Ted: "As a bus driver. Right."

"Yes sir. Because this line winds out being relatively heavy capacitance load."

"Sure."

"And we took it and broke it up. That bus I drew if one line. In reality it's not. It turns out to be the one that has the most inputs on it was four. So you had to take that bus and make it a Christmas tree to get in there. And this one was two. A two-way bus the other one was, so, I mean it's a two-way tree to get in. And in doing that way you use power, and then of course you use, I believe these are two-way trees. I'm not sure about those. But, never-the-less, it's those kind of things that begin to chew up your power."

"Mmm-huh."

"Then you do an estimate on your adder, which is relatively small, by the way."

"Sure."

"That's trivial."

"Your ALU did what sort of functions? Did it compare, add, and subtract?"

"Yeah."

"Did it do any shifting?"

"We did shifting outside the ALU. Not in that, but down here in our corrector before we did that. But, it would also do the logic the and/or, exclusive or, of course and invert."

"Sure."

"And, over here, we didn't have shifting."

"Here's some places where it had 8-bits of arithing. You actually would have folded that in the ALU?"

"Yeah, that is in the ALU, yeah. It's cheaper and fundamentally, we didn't want to have to load that bus any more."

"OK."

"It would possibly not have cost any more to have done that, but it's actually cheaper to do it this way. It's just that once you've got it to where it'll add and subtract, or add and complement/add, it's very simple to get the rest of it."

"That's an interesting difference between INTEL, this is, actually I added, before I used to go and fold the stuff in. A designer that we work with at INTEL and said, 'Take it out.'"

"Take it out, I keep it out."

"It's just stylistically the way you do it."

"No, it's easier to debug."

"Ah! It's easier to debug! That's why he does it."

Clash: "Yes, sir, oh yeah. Yeah, it's easier to debug. And I sympathize with him."

Ted: "OK, so that's why he does it. Interesting."

Ralph: "How much, you know, it may be a general question not related to exactly this, how much is, you know, when you bring up INTEL, I think there's another difference in that we're in the BIPOLAR. We are heavily into the gate array, and they're in the MOS design."

"Oh, yes, by the way, this is gate array."

Ted: "Sure, OK."

Ralph: "Obviously ..."

"So this is a gate array design. So what you have from INTEL is basically a MOS design?"

Ted: "In my thesis I'm going to be talking about this and where I find differences between the two, you know, all I can see, actually, in an issue about knowledge acquisition, multiple expert knowledge acquisition is a real hose, I mean because experts have differences of opinion. And that's good. They're supposed to."

"Sure."

Samit: "They have their biases."

"Well?"

Samit: "And their value systems."

Ted: "Sure. And one of the parts of things that people keep on trying to do is say well how do you get multiple experts to agree on something. I think that's totally wrong! You don't want them to agree."

Samit: "No, no."

Ted: "Because they're different. They're supposed to keep their different things in so you can have different rules for ..."

"... different rules for different parts of the thing."

Samit: "Do you also, shouldn't you also have the similar assumptions?"

Ted: "Sure."

Samit: "Why don't you try to combine them together to have a set of assumptions, and maybe once you understand the assumptions, then maybe you can find some similarity ..."

Ted: "Right, right."

Samit: "... you know, between their value systems."

"Well, that's, yeah. I design differently depending on what I perceive to be the criterion that I'm trying to satisfy. Like this was when I first forayed into the smaller stuff since, oh, 25 years."

Ted: "What have you done over the 25 years? You mentioned ..."

Clark: "I worked on the 701 which was the first one we ever built."

Ted: "Really, I've got a plaque on my desk of a little, you know, the plaque of the little plate from the 701. Yeah."

"I came up here in 1951 or 1952 or so."

"Wow, that's a really neat machine. We had that at the University of Michigan when I first got there."

"Yeah, well this would roughly have been, what, ten times or so or more? Not quite."

"So you worked on the 701?"

"The 701, the 702. I worked on all the 700 series and taught 701 and 702. And, let's see, the 7074 was the first one that I managed."

"Uh-huh."

"And I did the model 50. Actually I had the channels for the model 50, but the manager was never in town, so I did his."

"I've done that before."

"And I did FAA --"

"Uh-huh."

"... the FAA system -- you may not be familiar with that one."

"I've never heard of that one before."

"Perhaps, let's see. That's a 7-way multiprocessor into 12 memory boxes with 160 gp lines."

"Wow! You've designed a lot of control."

"I did that 20 years ago almost, and it's still in use, by the way. Any time you fly, you're in my system if you're in the continental United States."

"Really? Neat!"

"Or in London."

"Uh-huh."

"If you go into the field of air. That was the most fun out of anything I ever did. And since then, oh, I did the chip and I've done some architecture. I did the architecture for the Model 67, part of, I can't take credit for all that. I did the relocate. So I've worked with that sort of stuff. It depends on what it is you're trying to do. This case, I was trying to push gate array. My goal was to find if our gate array tools would handle 5000 circuits, and the vehicle I chose was a 370 Data path."

"Sure, sure."

"OK, now, that was what I really set out to do, was to see if our tools would push on out to there and turns out they will."

Ted: "Uh-huh."

Claud: *"With a little bit of changes made."*

"Well, there's always debugging."

"Yeah."

"When I first started running the 370 through, I had run 6502 designs, PDP-8 designs. I thought, hey those rules are solid."

"Right."

"No problem."

"That's right."

"It's about a third of the way through and all of a sudden, it goes into an unusual loop."

"Excuse me?"

"I had never seen this before."

"Yep."

"Uh-huh. And while I did a little bit of quick debugging and right, uh-huh."

"Yeah."

"Tools are interesting when you push them ..."

"Yeah."

"... you really push them."

"Well, we blew most of the tables because we had more data than they were used to seeing."

"This was a name stack table that blew up."

"Yeah. And so, with the help of the EDS people, we modified, and got it going. They finally wired it. Now what else can I help you with here?"

Samit: "Would this be a convenient time for a coffee break?"

"Right now!"

Ted: "That sounds good."

DJ Back To Critique

"By the way, while we're on that, if you, uh, find things that detract from your performance ..."

"Uh-huh."

"... often times in looking at machines that were not designed for 370, trying to simulate 370, it is the condition codes that really nail them to the wall."

"Why is that?"

Clark: Well, they're a bit awkward in 370 compared to some of the others, apparently and they can't get the bits over where they want them, and so on, and they waste an awful lot of time, of shifting them about. So, if you, you might look at a VAX; I've never looked at a VAX, interpreting a 370 and see what the condition codes and that sort of stuff is a pain in the neck for people to handle. And so, in designing these, when you get down to the hardware part, you always hold condition codes where they're readily available, because you can't afford the cost of those extra cycles."

Ted: "Yeah, that's actually something very interesting, or something I want to ask. When you have something like a PSW which has a whole bunch of junk --"

"Yeah."

"... it really, is in a sense, is a whole bunch of separate registers; one-bit registers."

"It is."

"And, uh, when you're dealing with a design, do you actually think of it as separate one-bit registers?"

"Yeah."

"Yeah, cause --"

"We conceptually say, PSW and short as a register only so that we know we're talking about a system status --"

"Yeah."

"... fundamentally."

"You may be amused to see, let's see, where's the, uh --"

"But, it's a lot of unrelated data --"

"Yeah."

"... is what it boils down to. It's only related in the context of defining the system in a period of time as far as the rest of it being related, it's not."

"This is the real output from my system rather than this drawing --"

"All right, you may have to show us how to read what you've got."

"This is actually the PSW module, I know cause that's register 14."

"OK."

"You can see that it's, you know, uh, different fields of it are getting loaded. All kinds of --"

"Yeah."

"... here's an input, we're getting fields from all over the place."

"Mmmmm."

"Different ones like here's a couple bits here going to bits there, here's uh, you know, 15, 16 bits there, 7 bits, there's 18, you know, and the output of this mess, look at this. Good

Lord! There's two bits go there, there's the instruction register product stuff ..."

Claud: "Uh-huh."

Ted: "... that goes out there. There's one bit there; there's another bit here."

"Yeah."

"There's another, you know, so ..."

"No, it's really ..."

"In terms of how to read this, module name is a register uh ... Register 14. Right."

"A port it's an output port; it's 64 bits wide and that port has different connections to it, of bit-width connection 55 to 52, goes to this other input mux and it goes to that specific port ... So you can see that those four bits go to those. And then down here, I have references back to my original ISP."

"The PSW is uh ..."

"That's a real ..."

"... Yeah, it gathers data from everywhere."

"Then the other output ... This here would be compiled into your stuff for ROS; this is symbolic ROS in a sense."

"Oh, OK."

"Uh, for control step two to two, on this particular VT body, I don't know what happen to the top of this printout; I quickly grabbed this up; these lines have to be active, so the FWRITE from there, and those mux ports have to be active. So, I know what ports and what lines have to be active for each one of my control steps."

"OK, then you know what bits to put on."

"Sure."

"You can take this and translate that so that it will actually cut ROS for you?"

"Yeah, there's a guy by the name of Nagle who has a program that takes similar ROS stuff and not only compiles it into ROS, but optimally choses the width of the ROS, and the depth of the ROS."

"Oh."

"... so that's kind of cool."

"Um-hum."

"That's what he did for his Ph.D, and there is someone who is now taking that stuff and extending it so it can have multiple controllers."

"Yeah."

"Uhm, which you know, I don't know how to do that. Most microprocessors have one controller and that's hard enough ..."

D.3.1 Tape change.

Chuck: "That takes an awful lot of technique."

Test: "Sure."

"And, uh, I guess I hold the patent on one that we used in 7074 back in '59."

"Uh-hum."

"We did, but that was a decimal machine, which I don't, well, we did a 2 out of 5 code. I don't understand why we did that. Well I know why I did; it's because I was designing a machine that was compatible with another machine that was 2 out of 5 code that was generated by someone ... who didn't have a full understanding of coding. But, nevertheless, I would do a multiply in a machine of this power; I would actually do multiply rather than the addshift. Addshift is not bad, but it doesn't cost that much to make a multiplier. Now, this one, you say, 'Controller - zero.'"

"Those are my inputs, to, like you had your ROS, and you had some inputs you had to bring over. OK, those are the inputs that I bring over, to my controller so that we can look at and make certain decisions about ..."

"Oh."

"... what goes."

"Oh, OK, all right. Then these, in essence, are modifying conditions ..."

"Right."

"... for my control."

"Like you can see that this, AND of a couple of lines that came over go into the controller."

"OK, uh-hum."

"Let's see, where do those lines come from? Those lines come from probably the instruction register. Yeah, those, oh, those are coming from the memory; wait, let me follow it over right."

"And you uh ..."

"Yeah, from the instruction register, is an or of a couple of things, will when I said bundle, there's actually two separate bunches of wires that come over from the instruction register, or they go into the controller."

"Our symbol for that is a slash ..."

"Oh."

"... and a number."

"Can you just draw that on there?"

"Well, we'd do that. I'd show you that -- now how many lines?"

"There's actually two lines, both of which were 3 bits long, I think, two bunches of things."

Claud: "All right. Two of 3 bits, you say?"

Ted: "Right, oh, OK."

"Normally you just put it on a three; you wouldn't put bits."

"Right."

"... but identify which is, that you have 2 three-bit lines."

"OK, great."

"You just draw a slash on it and show you that it's 18 lines."

"Sure."

"... or whatever and just put down slash-18 and that tells you how many lines you got."

"Yeah, that's a nice notation."

"It's more definitive than bundle."

"Right."

"Well, I must admit, some of these things get simplified for drawing out ..."

"Yeah."

"... cause, you know like ..."

"It's easier to draw than; it's just a slash and the number on it. And I don't know where we picked it up, but we're not the only ones; it's sort of in the industry."

"Oh, OK, great."

"OK, controller, and all right. That, in essence, is modifiers to my control."

"Um-hum."

"... and now this does compare, minus greater than ..."

"That's actually shift to the right."

"Oh, shift right."

"Right."

"Shift right and add. All right. This one is compare, add and subtract."

"Mmm-hum."

"OK, now, can you do successive addshifts here or multiply, although you've got a multiplier?"

"Yeah, I don't ..."

"Some designs can concatenate ..."

"You can actually concatenate through, um, if you're doing, uh, the stuff with the address ..."

"Mmm-hum."

Test: "... you get to the point where you concatenate through two of these to go back out."

Class: "Right."

"But, I don't need that much in terms of add/shift because of my multiplier."

"Yeah, well, you multiply. By the time you get it in, you may have to add/shift all inside that box."

"Sure."

"But uh, yeah, all right. OK, by the way, normally if you concatenate adders, uh, you're clocking gets a little more complex, or you loose speed."

"OK."

"All right, because you don't need it very often, and it's easier to take the extra cycle, most of the time, then to mess up your clocking."

"Sure. Control is such a bug-a-boo that you really try to simplify that whenever you can."

"Yes, OK, now I see something. Oh, this is just a bucket of constants."

"Yeah."

"OK, yeah, you always need those, a lot."

"Do you have that normally in ROM or do you just normally burn that on the chip or do you use multiplexors to generate it, or how do you normally generate your constants?"

"It depends on the designer."

"OK, yeah."

"I have my preferences."

"What are your preferences?"

"Well with me, I go for the flexibility. I very seldom burn them in unless it's ..."

"OK."

"Well, I have done more flexible designs than I have really honed for speed."

"OK."

"You'll burn it."

"OK."

"OK, if you really are honing it for speed, like building vector processes, which I've never built, although I've consulted with Lowen, but with that you'll burn them in and all, because that is all -- out speed -- forget everything else, including cost almost. Now, these are a set of ..."

"Those are a set of single bit registers, uh, FEW what are they? Excuse me."

"Those things, they always focus at the wrong distance."

Ted: "Guaranteed."

Clau: "I can see close-up. It's reading signs down the road that I have a problem with."

"Those are error flags from the floating point information."

"Oh, OK, so these are my error flags. All right, I can ..."

"For floating point."

"... for floating point."

"Right."

"... and I presume ..."

"That would be my channel instruction register, but I didn't uh ..."

"OK, so you need those and hardware, no question, and, OK, channels."

"Those are my floating point registers."

"Yeah, OK, and this is a control register."

"Control registers."

"That's 16. This is just standard register, OK."

"MB is my primary memory."

"OK, all right. Now these represent ... You've got to get unaligned. Now this is a function for zero to 63."

"Right, especially there's some functionality about doing calculations to get some unaligned stuff and it will buffer that into a register. The reason why it buffers that into a register is because of the debugging ability. I want to be able to see if that functionality works properly. The real reason why there's a register there is because it goes over a clocked boundary I needed a place to put it."

"OK."

"... and that was the temporary register. I named it that so I would remember what it was."

"All right. In order to get the stuff on boundaries, why we have to do some shifting or cross-gating, or whatever."

"Yeah, because you don't know whether it's a boundary basis to start with ..."

"Yeah, it may be bits, and so you ... all right. This is for the same reason."

"Right."

"So, fundamentally or ..."

"To get and put for the PSW you do the same thing because the PSW is such a funny thing."

"Yeah, it's a lot of pieces, so you ... So those almost would be part of controls except that they're handling data ..."

154

Test "Right."

Client: "... the object of some very complex control. And this is your first byte?"

"Actually that came about because the way the description was written, it was decided one would use a function in ISP, which said return the first bit that's a one and how many bits over is it?"

"Oh, OK, oh, this is your first one, so you're going to count and tell me ..."

"Right."

"... the bit number of the first one."

"Right."

"OK, fine."

"That's the result of that."

"Got to have that. And uh ..."

"And that's the interrupt vector."

"Yeah, OK, and then we come to the instruction register, I presume."

"Right."

"And let's see, R04 three-two bitter."

"That has to do with the memory management stuff, as I recall. Let's see that has a segment table ..."

"OK."

"... and 32 bits a segment table."

"You got another one here."

"There's a page table."

"A page table."

"Right, and a segment index."

"Index."

"A page index, page displacement."

"Displacement."

"Right."

"And now we come to floating point."

"Now we start dealing with floating index stuff."

"Floating point stuff which uh, ..."

"Then there's the DAT box return itself."

Clad: "Mmm-hum, DAT return, OK."

Ted: "All right, in terms of temporary registers, I end up having this temporary register and that temporary register between the two of them, in your structure, you would bring two buses into here."

"Yeah."

"What I did, basically is, I had one bus coming into here, and a bus coming into this which went into there."

"Mmm-hum."

"So like, if we follow this crazy line around --"

"OK."

"We find that goes into the bottom of the ALU. This line here is that --"

"Yeah."

"-- let's trace that down. So that's --"

"Oh, well, then maybe I ought to show you more on mine over here while I'm at it."

"All right."

"I had an AB register here."

"AH."

"Now, let me tell you, most of the time I would hold that gate down and read straight through it."

"OK."

"But if I'm clocking, I need it to hold at that point."

"Right, sure."

"But mainly -- let me tell you what I use it for -- was when I would clock here is so I could drop these lines --"

"Right."

"-- and use them for something else."

"Right, right."

"And this would still hold that through the adder so I could be latching."

"Right."

"That allowed me to latch, to read from, OK, my clock looked like this, that's not true, it's a balanced clock, but nevertheless I could read from, I could read on that edge, and I could write back in --"

"Mmm-hum."

"-- later, onto that same clock --"

156

Te: "Mmm-hum."

Clark: "... provided I had a buffer."

"Right."

"So, I made it a rule never to store in the capacitance. So that's the same thing."

"I don't store in capacitance either."

"I don't believe in that."

"I think it's a crock."

"Well, that's where you get intermittence."

"That's right. You get intermittence. Also, you're designing the chips manufacture, so you're in variability there."

"Yeah, so, when I find anybody doing that, I write them off as a designer."

"That's right. That's what the purpose of that was."

"OK, well, when I saw that, I realized I didn't show you those, two ..."

"OK."

"And that's what they're for is to hold it. And we have some other hold registers, I don't remember exactly where; my MAR, Memory Address Register, is on this 24-bit line. In reality, if you read into that, and come around, but functionally it's the same. I won't go into why we do that; it was for checking reasons, I can tell you. Yeah, by the way, over here, this actually goes up to this, and then we read out and back down. But that's so I can use the parity checker."

"OK."

"Functionally, it did nothing for me."

"OK."

"All right, I guess we've about almost finished, haven't we now?"

"Yeah."

"We've done that and this, they're buffers, and here's our prefix."

"The prefix, the CPU timer, and the clock and the time of day clock."

"OK, that's just the architectural stuff that you've gotta have."

"You had those out in the stack also?"

"Yeah, I showed them out; I just didn't have room ... And there's the signal out here."

"Signal out, I/O done."

"There's a one-byte job's."

"Yeah. It's R-67 here, you're indexed that way."

Ted: "Yea, I am indeed am; this is high-class operation. Written out by hand is not as high-class as I would like it. But R-67 is a signal output for direct I/O. That's, uh ..."

Claud: "Oh, OK."

"... ten bits out. So you had eight bit out ..."

"Yeah."

"... and I had ten bit out."

"Yeah, all right."

"I had eight bit, the next I/O data register. That's eight bits ..."

"Yeah."

"... and then the next up is external-register; bit zero is the timer interrupt; bit one is the console interrupt, uh ..."

"Mmm-hum."

"... next up is device register; that holds the device address as zero to 255."

"OK."

"Then the channel address, channel instruction and channel condition codes ..."

"Yeah, OK, that's all in my channel stuff."

"Right, that you had out in the architectural register?"

"Mine was out, uh ..."

"On a separate chip."

"This chip again in a different implementation."

"Actually, I have the design for that stuff, but I decided not to draw it because I didn't see it anywhere in your stuff."

"No, it's not. No, I didn't show it, no. What's this?" Oh, that's a channel interrupt?"

"Interrupt, or instruction, I don't know. It must be interrupt."

"Instruction, I'm sorry. Is that 62?"

"It's 32."

"Yeah, it's channel interrupt."

"Channel interrupt."

"Request."

"And there's a PSW and PSEC."

"It's 31. That's the extended code stuff."

"I've got ya, yeah."

158

Test: "Where'd you do your extended code stuff?"

Chase: "Eh, yeah we handled that out of board. We had that, there was actually two registers. Yeah, we carried two registers, that was in the stack."

"Right, OK."

"Memory buffer registers, OK."

"Buffer, double-buffer, and address register."

"Yeah, um-hum, OK, I had those um, on memory address register."

"By the way, there was a ROS address down here, too, if you want all that."

"Sure."

"We have, there's a ROS data register, ROS data register there, that was on chip, and there's a ROS address register that came down here as well. And, let's see, there was a comparator ..."

"The ROS address was also coupled to your ..."

"Oh, it had all these bits and so on, and the ALU; there's a whole complex mess, uh, because that is a critical path for speed."

"Yeah."

"And so you'll find that we bypass some of that to get free-charging on lines and all that sort of stuff. But that's uh, technology dependent."

"Sure."

"I don't consider that as being ... that's the tough engineering part. That has nothing to do ..."

"Yeah."

"... with architecture."

"Now, we also did our DAT compare right here; I don't remember whether I did it in this, I did, yeah. But I had a DAT compare function here, only to compare. The rest of it, I just didn't have space."

"I see. You put your compare right there, OK."

"Yeah. Uh, and then the rest of it was done in microcode, simply because we ran out of blocks."

"This is, could you do me a favor and plug it in that plug over there? In talking between the two, I don't want to lose half the conversations."

"Yeah, my voice carries, and normally ..."

"Mine carries; I think this is a pretty good tape recorder in that the microphone was a good design."

"Um-hum."

Ted: "And uh, I've had good luck with it so far."

Clark: *"It looks like a nice one. Uh, I don't know much else about that. I think I've pretty well covered the little bit I had."*

D.4 Summary

"Well, let me try summarizing."

"OK."

"... some of the key points. Uhm, one is that in general, the registers I have here, you indeed have them mostly out board on the stack."

"Right."

"Uhm, you have primarily an 8-bit data path with several registers to go through for reasons of cost and there wasn't an increase in performance. Uh, you have the single ALU; I have multiple ALUs. Both, again because it's a Cadillac version, more higher performance."

"I do have a multiple; one for my address."

"Yeah OK."

"I have an address, which is a 24 bitter, and then an 8 bit."

"And I have three, so, I have one more."

"Yeah."

"You fold your uh, logical operators into your ALU."

"Right."

"... and you mentioned that they possibly would have it out for debugging, but you like it in for cost reasons. OK?"

"Correct."

"Uh, let's see, uh, constants are the same; so pretty much of the design is pretty much the same sort of design as yours, with exception of the bit width, uh, being wider on mine than on yours."

"Right."

"OK."

"You may rearrange some gating between the registers at the time you start laying out your busses."

"Sure."

"But that is technology dependent."

"Sure."

"... and so you don't make that decision."

Test: "When you said you had a Christmas tree four-part bus, what did the bus actually look like?"

Clark: "That was for driving capability and it looked like this."

"OK."

"OK, I had an OR at this point — an OR circuit — and that was really the bus, OK."

"Uhm-hum."

"This is your bus; we're sure of how to get there. And then I would take this many, well, let me see. I guess I had a driver, and it drove into that. I had four of these."

"Uh-huh."

"Uh, excuse me, not so — that's the driver down below. Let me get this straightened out; I would take part of my registers — you can only do — and I'd put 'em in this way. My next set of registers, they're here; would go into the next one, and uh, so on, and then I'd have four of these lined up to take care of my 17 registers."

"Sure."

"OK, actually, I don't want to put 16 in, so I divided it into four fours is what it boiled down to. That's one, two, three, four; one, two, three, four; one, two, three, four."

"OK."

"And so I would handle 16 registers, and these then, would go through the OR, and it was all for powering reasons like that."

"Right."

"And then this turned out to be the bus. The others would be, uh, where I was using only two of those, and then to an OR —"

"Um-hum."

"... and this would be for your registers. And that's the way we did that."

"OK."

"Uhm, let's see."

"That's what I meant by Christmas tree."

"Sure."

"And that was for powering and to break that capacitance up into pieces."

"What is your overall impression? If I were a designer coming to you with this as a first-shot at a design, and saying that my constraints were for a high performance machine, uh, would you think this is an OK, machine?"

"It's a good starting point because ..."

"... you can't go much further without knowing your technology."

"Sure."

161

Claude: *"Technology, your package, and decide uh, uh, then, because, you see, take this Christmas tree thing over here."*

Ted: *"Yeah."*

"That doesn't show up logic-wise at all ..."

"Sure."

"... anywhere. That's just a straight line."

"That's right."

"And then you've got to add all that in, because of the technology I was working with."

"Right."

"And so, that'll work."

EXHIBIT

36

The VLSI Design Automation Assistant: From Algorithms to Silicon

T.J. Kowalski, D.J. Geiger,* W.H. Wolf, and W. Fichtner
AT&T Bell Laboratories

The complexity of integrated circuits has increased at an enormous rate, from a few hundred components on a chip in the 1960s to hundreds of thousands of components in the VLSI chips of the mid-1980s. This explosion in complexity was made possible by advances that continue to shrink the dimensions of devices in IC fabrication.

During the last few years, we have witnessed the realization of complete systems—32-bit microcomputers¹ and digital signal processors,² for example—on a single VLSI chip. While these designs were realized with conventional processes in two-micron design rules, considerably larger sys-

tems will be possible with state-of-the-art one-micron and submicron silicon technology.

Although we can build complete systems on a single silicon chip, the design of a VLSI chip is extremely expensive. It can take several years to go from the conception of an architecture to the completion of the final product. The complexity and expense of chip design makes it increasingly difficult to build cost-effective, low-volume, special-purpose VLSI systems.

We believe the only solution to this problem is to build new CAD tools that are capable of automatically performing more of the synthesis process. In the past few years, tools have been developed to act as expert assistants for stages in the VLSI design process, from architecture definition to layout. Our approach seeks to aid the designer with tools that automatically produce data paths and control sequences from an algorithmic system description within user-supplied constraints.

An automatic design aid offers several advantages. First—and probably most important—it drastically

Summary

Our ability to fabricate complex VLSI chips is outstripping our ability to design them. To reduce the design time for complex chips, we are creating an expert design assistant—programs that automate as many of the design tasks as possible while allowing the human designer to control the synthesis process. A series of programs translates an algorithm into a full layout. Some programs are expert systems, while others use traditional algorithms.

Our early results lead us to believe that a knowledge-based expert system is an appropriate tool for bridging the gap between logical and physical design.

*Geiger is now with Carnegie-Mellon University.

decreases the time it takes to design a chip. Second, it allows the designer to investigate a wide variety of different strategies before deciding on a final design. This can even include the evaluation of different technologies in order to choose the one best suited to the particular problem. Third, it allows designers to use the most up-to-date fabrication technology available.

A cornerstone of our hardware-synthesis approach is the use of knowledge-based expert systems. Such systems make decisions based on knowledge, expressed as rules, obtained from expert designers. This approach to hardware synthesis has several advantages. First, much of the problem is too poorly understood, and the nature of the problem changes too rapidly to lend itself to solution by traditional algorithmic approaches. Second, a knowledge-based expert system can replace expensive backtracking with knowledge-directed search, thus producing good hardware designs in a reasonable amount of CPU time. Third, the organization of an expert system makes it easy to add, modify, and test new expert knowledge. Finally, the discipline of encoding expert knowledge into rules enriches our understanding of both the knowledge and the structure of the problem.

Therefore, while some stages of chip design—notably layout generation—can be handled well by algorithms, the most challenging areas require the application of knowledge-based expert systems.

While our research is not complete, we have promising results at both the high and low end of the chip synthesis problem. At the high end of the design process, we have demonstrated that a knowledge-based expert system can produce quality block-diagram designs from algorithmic descriptions of chips.³ At the low end, we have built on the work done on symbolic layout⁴⁻⁶ and on our work in compiling layout cells from simple descriptions.⁷ This experience leads us to believe that a knowledge-based expert system floor planner can bridge the gap between the high and low ends of chip design.

In this article, we describe our ap-

proach to automatic design of VLSI silicon chips from algorithmic descriptions, briefly review related work, and present our plans for future work on the system.

Methodology for hardware synthesis

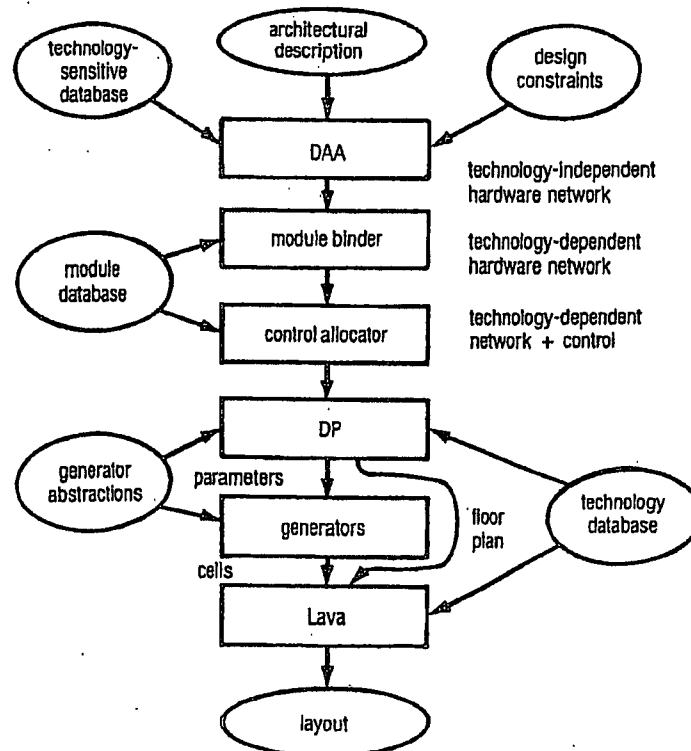
We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS.

This algorithmic description is then transformed into a technology-independent network, built of modules of medium complexity—registers, adders, and multiplexers (MUXes)—and a

symbolic control description. This network is used to synthesize a technology-dependent network that includes implementations of the technology-independent modules and the controller in the target technology. The next step is the construction of a floor plan that describes the placement and wiring of the chip in terms of the modules in the technology-dependent network. The floor plan guides the assembly of the final layout of the chip.

This design process is illustrated in detail in Figure 1. Typically, the chip proceeds through six different tools as it is transformed from an algorithm to a layout. Briefly, the steps in the synthesis process are:

- The design automation assistant (DAA)⁸: a knowledge-based expert system responsible for transforming the algorithmic description of the chip into a block diagram and a symbolic control specification. In addition to this algorithmic description, written in



SP03988

Figure 1. The VLSI design process.

ISPS,⁹ the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs.

- The module binder¹⁰: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either selected from the module database or fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University.

- The control allocator¹¹: an algorithmic program that designs the controller for the data-path logic designed by the DAA and the module binder. It can target the controller toward either a microprogram engine or programmable logic array (PLA) implementation. The control allocator was developed at Carnegie-Mellon University, and work is being done at AT&T Bell Laboratories to add pipeline-style designs.

- The data-path floor planner (DP): a knowledge-based expert system that is responsible for the complete physical design of the chip. It arranges the modules selected by the module binder and control allocator into a data-path floor plan and creates the cell-layout parameters. The floor planning process is constrained by user-supplied restrictions on area and speed and on knowledge supplied by two databases: the technology database and the generator-abstraction database. The technology database provides generic information such as unit-wire capacitance, while the generator-abstraction database provides estimates of the size

and speed of particular cells. The DP is in the early stages of development.

- Generators: algorithmic programs that produce technology-specific custom cell layouts from parameters that describe logical, electrical, and layout requirements. Generators are not implemented as knowledge-based expert systems, but instead are written as algorithms by layout experts. Sample generators and programs exist to compile cells. Current work focuses on understanding the representation of generator knowledge, and expanding the library of generators.

- Lava: an algorithmic program that compacts a layout for each individual cell, then assembles the cells

While some stages of chip design can be handled well by algorithms, the most challenging areas require the application of knowledge-based expert systems.

into a complete chip, stretching them when necessary to make the required connections. The result is a layout that can be used for fabrication. Lava, initially developed at Stanford University, has been used to design several chips.

Related research in synthesis design

The work done thus far in digital design synthesis is reminiscent of the early chess playing programs. While it is easy to get a computer to play a legal game of chess, it is exceedingly difficult to get it to play a good game of chess. Many design-synthesis projects have, like the early chess programs, produced a working result,¹² but the quality of the results leaves something to be desired. Some programs have produced designs of unacceptable quality, others are too expensive computationally for even small designs, and still others tackle only a small part of the VLSI design problem.

VLSI synthesis research can be divided into several major categories. One category is logical design—the synthesis of a technology-independent hardware description from an algorithmic specification. Research efforts in this category include a classical compiler,^{13,14} an interactive optimization system,¹⁵ a constraint language system,¹⁶ a graph-theoretic system,¹⁷ and an iterative heuristic approach.¹⁸ These methods have various advantages and disadvantages, but in our opinion none of them produce acceptable results on realistic examples in a feasible amount of CPU time.

At the opposite end of the synthesis spectrum is physical design—the production of a layout from gate- or module-level hardware descriptions. Polycell systems like LTX II,¹⁹ and standard-cell systems^{20,21} can produce layouts with little or no human intervention by placing and routing pre-designed cells that implement logic and memory functions. This category also includes module generators—programs that produce related cells according to the designer's specifications. Module generators include layout languages embedded in programming languages (Chisel),²² compilation algorithms (such as Dumbo⁷), and knowledge-based expert systems.^{23,24}

Between these two extremes is the silicon compiler. A silicon compiler takes as input an algorithmic description of a chip, and produces a layout. The compiler does this by using a target architecture for the logical design and a target floor plan for the layout design. The first silicon compiler was Bristle Blocks;²⁵ another well-known example is MacPitts.²⁶ In both systems, the logical and physical designs of the functional units are predefined by a human designer.

Design strategies

Our strategy is to break up synthesis into several programs. We believe our approach has some advantages for software engineering: it simplifies the implementation of the subsystems; it

makes it easier for the designer to examine the state of the design and to modify it if necessary; and it allows the choice of different computing environments for different levels. While non-algorithmic programs such as the DAA and DP are best implemented in the environment of a knowledge-based expert system, algorithmic programs such as Lava and the generator are best implemented in a programming-language environment.

We have two reasons for choosing algorithmic generators over a general knowledge-based expert system approach. First, we can write specialized generators corresponding to the hardware building blocks—ALUs and registers, for example—and make use of expert knowledge about their physical design. Second, we believe that knowledge-based expert-system layout generators cannot produce layouts for high-performance modules with the same success as has been achieved with algorithmic generators.²⁷

DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize an architectural implementation from an algorithmic description with constraints. The description is written in ISPS; a sample description for a Digital Equipment Corp. PDP-8 is shown in Figure 2. The first part of the description lists the current-page and instruction words; the second part of the description tells how to calculate an effective address. The DAA actually works from a data-flow representation extracted from the ISPS description. This data-flow representation resembles the internal form used by most optimizing compilers, and it is less sensitive to the writing style of the algorithmic description.

The DAA produces a technology-independent hardware network. This network is composed of modules, ports, links, and symbolic microcode. The modules can be registers, operators, memories, and buses or MUXes with input, output, and bidirectional ports. The ports are connected by links

and are controlled by the symbolic microcode.

The DAA uses a set of temporally ordered subtasks to perform the synthesis task. It begins by allocating the base-variable storage elements—constants, architectural registers, and memories with their input, output, and address registers—to hardware modules and ports. Then a data-flow BEGIN/END block is picked, and the synthesis operation assigns minimum delay information to develop a parallel design. Next, the DAA maps all data-flow operator outputs not bound to base-variable storage elements to register modules. Finally, it maps each data-flow operator with its inputs and outputs to modules, ports, and links. In doing so, the DAA avoids multiple assignments of hardware links; it supplies MUXes when needed. These last two mapping steps place the algorithmic description in a uniform notation for the expert analysis phase that follows.

The expert analysis subtask first removes registers from those data-flow outputs where the sources of the data-flow operator are stable. Operators are combined to create ALUs according to cost, partitioning, and pipeline estima-

tors across the allocated design. The DAA also examines the possibility of sharing nonarchitectural registers. Where possible, it performs increment, decrement, and shift operations in existing registers. Where appropriate, it places registers, memories, and ALUs on buses. Throughout this subtask, constraint violations require trade-offs between the number of modules and the partitioning of control steps. The process is repeated for each data-flow BEGIN/END block.

The knowledge-based expert system approach followed by the DAA uses a weak method,²⁸ *match*, in place of extensive backtracking. The DAA uses *match* to explore the space of possible designs by extending a design from an initial state to a final state without any backtracking. The DAA proceeds through its major tasks in the same order for each problem; it never varies the order and it never backs up in any problem. This means that at any intermediate state, the DAA can determine how to extend the design to achieve an acceptable result.

DAA produces a technology-independent, but technology-sensitive implementation of the ISPS description.

```
Example :=
Begin

**Storage.Declaration**

cpage\current.page<0:4>,
\Instruction<0:11>,
pb\page.0.bit<> := 1<4>,
pa\page.address<0:6> := 1<5:11>

**Address.Calculation**

Global eadd\effective.address<0:11> :=
Begin
Decode pb =>
Begin
0 := eadd = '00000 @ pa,
1 := eadd = cpage @ pa
End
End
End
```

Figure 2. Sample ISPS description.

SP03990

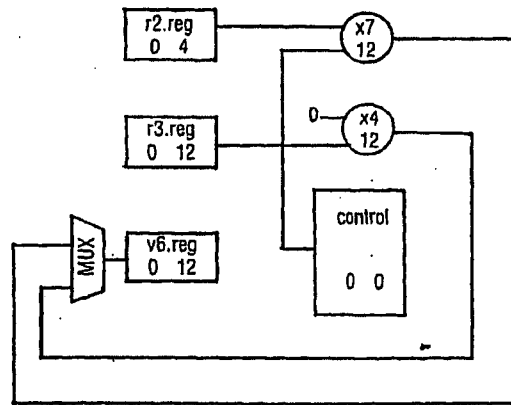
The technology-independent network and associated control are described in the SCS language; a graphical representation is shown in Figure 3. The circles concatenate two sets of signals, the trapezoid is a two-input MUX, the small rectangles are registers, and the large rectangle is the controller. Because the initial ISPS was so simple, the final design includes only some wiring, the specified architectural registers, a MUX, and the controller. The control specification is also included in the SCS description. The control is described in terms of abstract operators—control steps and selects, for example—that can be implemented in a variety of controller technologies.

Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint.

If the module database does not contain a cell with the right combination of functions, a combination of existing cells is used. For example, if the binder is unable to find a single module corresponding to some combination of an ADD and an OR, it will select separate ADD, OR and multiplexer cells to generate such a module. Similarly, if the module database does not contain a cell of the proper bit width, an existing cell may be replicated. Finally, if the DAA has specified the use of buses, the module binder will provide technology-dependent bus drivers.

Control allocator. The control allocator translates the technology-independent control specification, generated by the DAA, into a controller implementation in a chosen technology. A style-independent part determines the control signals that drive the data path and a style-dependent part designs the sequencer. This

Figure 3. Sample structural specification.



method is desirable because the style-independent part is the same regardless of the way the controller is implemented (PLA, microprogrammed controller, or random logic), but the style-dependent part changes with each different implementation.

The control allocator receives information from the technology-dependent hardware network and the module database to produce an output file for a specific style of controller. The structure-and-control-specification file²⁹ provides the control signals and the technology-dependent data path;

the module database provides the required control line values. Currently, our control allocator can produce PLA-style output for the Berkeley³⁰ and the AT&T Bell Laboratories³¹ PLA generation tools or a microprogrammed-style output for an AM2910 microengine.

Floor planning. The floor planner produces an abstract physical design for the technology-dependent network designed by the DAA and the module binder. The floor plan includes a placement and wiring plan for the

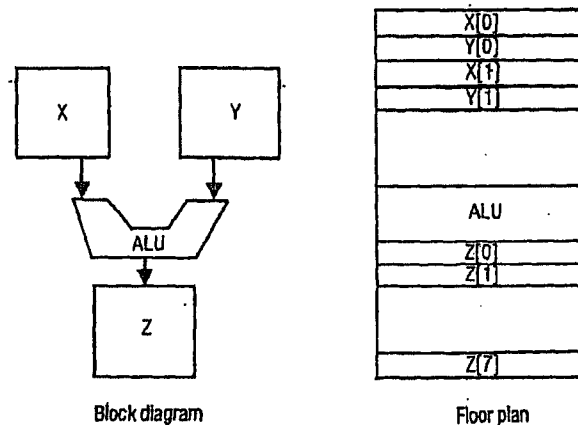


Figure 4. Sample netlist to floor plan transform.

modules and specifications for the design of the cells.

The DP knowledge-based expert system is driven by a set of rules that embody expert knowledge about floor planning. There are two categories of rules: those that build an initial floor plan for the chip based on the logical design and user constraints, and those that improve on the initial design.

The use of rules to build an initial floor plan can be illustrated using the technology-dependent design shown in Figure 4. The left side of the figure shows a block diagram for a system made of two input register arrays, X0-X7 and Y0-Y7; an ALU; and an output register array, Z0-Z7. The right side of the figure shows a common and efficient floor plan for this system: The input register arrays are interleaved above the ALU; the output register array is placed below the ALU.

This transformation from block diagram to floor plan suggests three rules: If word arrays of elements (such as X0-X7) are encountered, then rules are to be applied to each element independently; if elements are connected in parallel, then they may be interleaved; and if two elements are connected, then the source element should be placed above the sink element in the floor plan.

The second category of rules describes optimization methods, the type of knowledge we normally associate with expert designers. An example optimization transformation is shown in Figure 5. The critical timing path for the circuit flows from register B through the ALU and into the output latch. The initial floor plan, however, places register A closer to the ALU than register B. The transfer time from B to the ALU can be reduced by swapping A and B, thus reducing the capacitive load that B must drive.

A secondary source of expert knowledge, one not contained in the rule database, is the generator-abstraction database. Each generator is represented in the database by an abstraction that includes essential knowledge of the physical, electrical, and logical properties of the generated cells. This abstraction is necessary because the in-

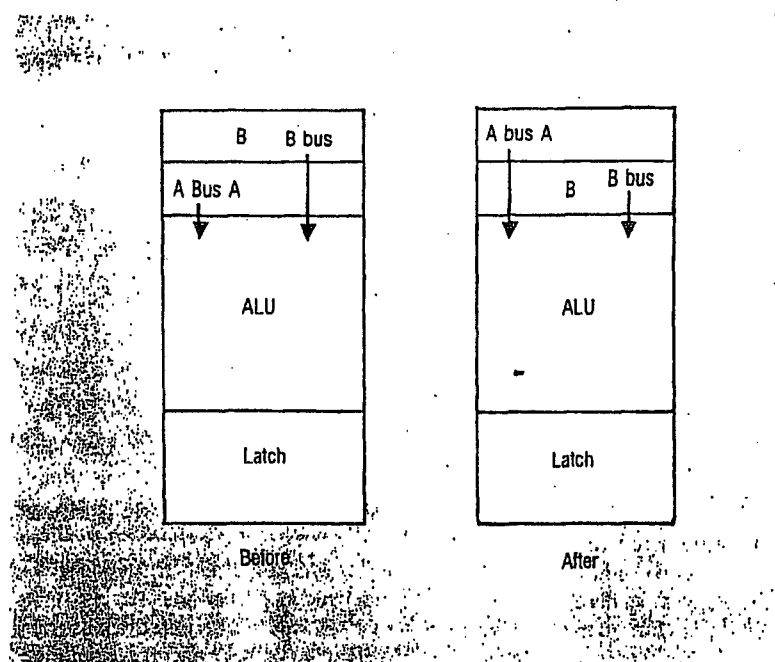


Figure 5. Sample optimization transformation.

formation is difficult to obtain directly from the generator. Much of the desired information—input load capacitance, aspect ratio, and drive current, for example—is included only implicitly in the generator code, so the floor planner has to make many decisions based on the information in the technology database. The desired cell

Using a series of algorithmic and non algorithmic techniques, our goal is to generate silicon from algorithms.

could be generated and then measured, but this would be too time-consuming for chip synthesis.

Some of the information in the abstraction is precise; other information is estimated from heuristics and particular knowledge about the generator. An abstraction may contain any mix of exact and estimated information, as specified by the abstraction's designer (usually, the generator

designer). The database includes methods for estimating physical parameters (area and shape, for example) and timing parameters (delay and drive current) from partial descriptions of cells.

A floor plan is derived from the module binder's structural description by constructing an initial design and then improving the floor plan. Both initial placement and improvement are driven by the rule database and are guided by information provided by the generator abstractions. At the same time the floor plan is being built, the specifications for the modules—size, shape, delay, etc.—are being refined. These specifications will be passed to the module generators to guide the production of the cells.

The resulting floor plan is described in the Lava language. A sample floor plan for a bit-slice design is shown in Figure 6a, and a drawing of the floor plan is shown in Figure 6b. The floor-plan header describes the chip's connections to the outside world. A signal can be a single wire, such as packet, or a bundle of wires, such as bitslicebot. (Actually, bitslicebot and bitslicetop are bundles of bundles.) The abut op-

erator, |, asserts that two cells are to be abutted together and stretched if necessary to connect all wires between them. The keywords left, right, down and up show the relative placement of cells. Wire routing is contained in cells that are abutted to the cells implementing the modules.

Cell generation and assembly. A cell generator embodies expert knowledge of how to design a family of cells. The

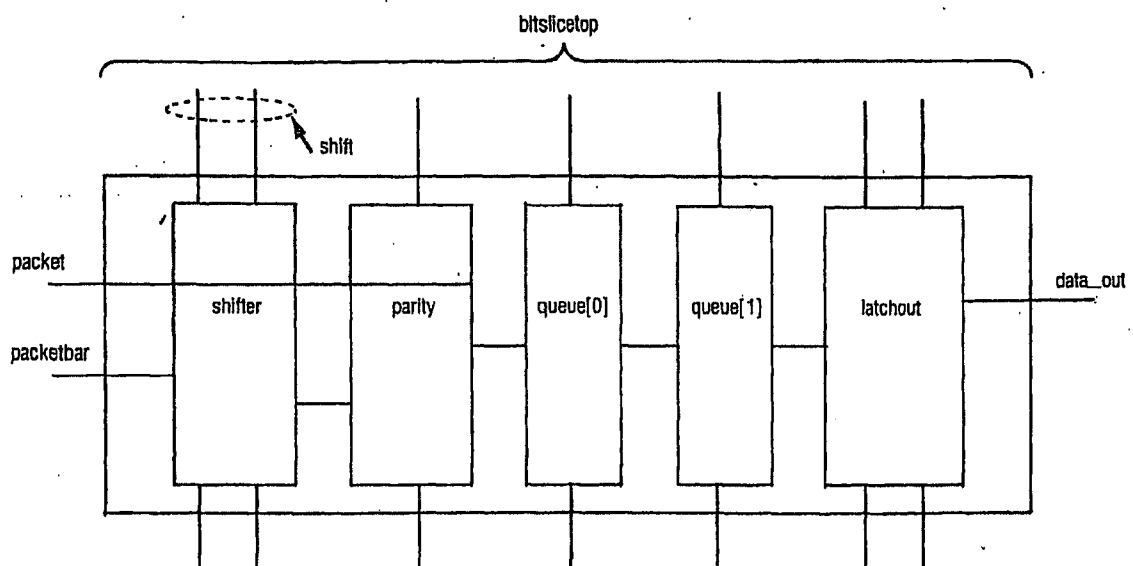
generator program produces a Lava description of a cell from parameters that define the requirements on the cell. At present we can generate modules from a database of predesigned modules or by compiling them using the Dumbo cell compiler. Cells can be added to the database via a Lisp-based symbolic layout editor.

Part of a cell-generator program is shown in Figure 7. This sample program produces an NMOS half-adder

circuit in a transmission gate implementation.

The Dumbo compiler works from a predesigned cell, defined as gates and transistors, an external pin list, and hints about the form of the cell's layout. A preprocessor customizes the generic description based on the input parameters by selecting transistor sizes, optional components, etc. Dumbo then uses placement, routing, and component definition algorithms to

```
cell bitslice(bitslicebot, bitslicetop, packet, packetbar, word_out) abut
bitslicebot : bitslice : down;
bitslicetop : bitslice : up;
packet, packetbar : diff: left;
word_out: poly: right;
{
  shifter(packetbar, packet, bitslicebot.shift, bitslicetop.shift) |
leftof
  parity(databar, data, data_out, bitslicebot.par, bitslicetop.par) |
leftof
  iterate NWORDS | right queue(queueleft, queueright,
    bitslicebot.q, bitslicetop.q)
    { queueright # queueleft } |
leftof
  latchout(queueright[NWORDS].shiftsig, word_out, bitslicebot.outputs,
    bitslicetop.outputs);
data_out # queueleft[1].datasig;
}
```



SP03993

Figure 6a. Sample floor plan specification.

turn the generic cell into a symbolic layout.

Layout representation. The Lava chip compiler compacts the individual cells and assembles them into a chip according to the floor plan. Lava is responsible for stretching cells to make the electrical connections specified in the floor plan; it can also optimize the cells to reduce the layout's size. Lava produces a symbolic layout with its components placed at their exact positions in the final layout. This symbolic layout can be macro-expanded into a pure layout that can be used to fabricate the chip. Figure 8 shows the XYMASK version of the layout for the NMOS half-adder cell.

Future work

We are exploring the problem of synthesizing a working silicon chip from an algorithmic representation of a VLSI system. Using a series of algorithmic and nonalgorithmic techniques, our goal is to generate working chips from algorithms.

This article has outlined the work in progress on several tools. The first tool, the DAA, is still being developed, but is solidly in hand. Like human designers, the DAA becomes a better designer as its rule memory expands. Currently, DAA is being expanded to handle pipeline-style designs. This work is in its early development stages, but shows great promise.

```

/*A*/
cell tgha (vddtop, vddbot, vsstop, vssbot, xin, yin, sum, carry) leaf
vddtop : blue : up;
vddbot : blue : down;
vsstop : blue : up;
vssbot : blue : down;
xin : red : up;
yin : red : up;
sum : red : down;
carry : red : down;

compactxy;
(p0 = point blue (1,20))
  (width 4) vddtop;
(p1 = point blue (22,20))
  (width 2) p0;

(p2 = load (2,2) total (6,1,16)) source;
  (green (p2) -> (load (1,5)))

p3 =
  (p1 -> (p2 -> (green (3,1) rotate (4,16)) then
    (red (p1) = point red (1,20)))

xin #
  (red (p3) = word (29,19))

carry #
  (red) 16 gate;

sum #
  (red) 11 gate;

```

Figure 7. Sample cell generator fragment.

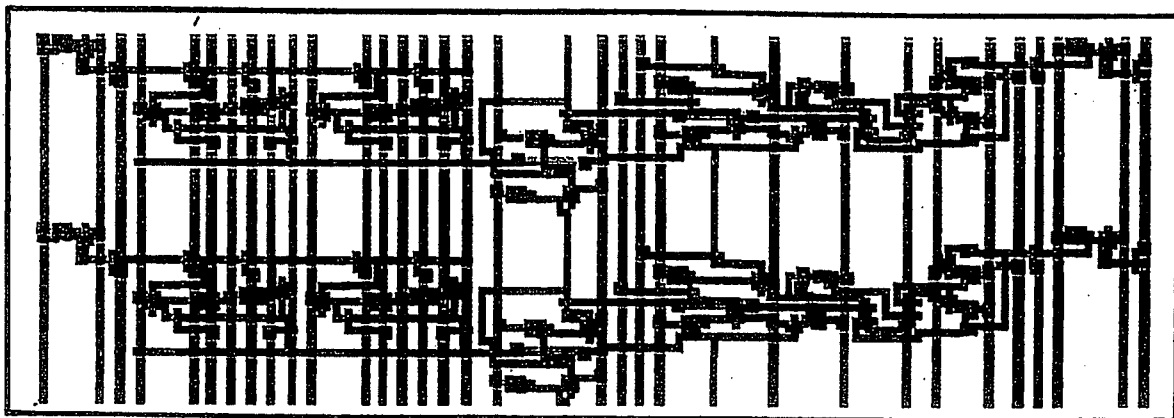


Figure 6b. Sample floor plan design.

SP03994

The module binder is nearing completion at Carnegie-Mellon University.

The control allocator is complete, and has been used to design several PLA and microcoded controllers. The largest controller designed was for the RCA 1802 CMOS microprocessor. Changes will be needed in the control allocator if it is to design controllers for pipelined systems.

Development of the floor planning expert system is in early stages. We are building a floor planning knowledge base from interaction with chip designers and analysis of sample chip designs, such as the FLASH signal processor and other chips proprietary to AT&T Bell Laboratories. We are building the generator-abstraction database from interviews with layout designers and analysis of existing generator programs. Currently, we are redesigning several chips with our synthesis tools and comparing the automatic and hand designs.

After completing some working chip designs, we plan to experiment with high-level optimization based on

layout analysis. Information about the physical design of the chip, gained during floor planning and cell generation, may be useful in optimizing the logic design or architecture of the chip. For example, if the floor planner is unable to produce a layout with an acceptable delay for a signal, the clocking scheme of the architecture may have to be redesigned. Automatic optimization is intriguing because the programs can generate and analyze much more information than can a human designer working alone, and because the new design can be reimplemented quickly. The problems are formidable, however, and our goals in this area are long-range.

This article has described our approach to automated VLSI design. We have discussed how we break the integrated circuit design process into stages and how to implement those stages as programs. We have aimed for a methodology that automates as much of the design work as possible, but still allows the designer to

control the outcome of the synthesis. We hope to free the designer to solve the important problems in a chip design.

We strongly believe that only by automating more of the chip design process will we be able to exploit the possibilities of advanced VLSI fabrication technologies. We also believe that the combination of algorithms and knowledge-based expert systems offers significant benefits when building computer-aided design tools for VLSI. Some synthesis problems are solved best by algorithms; others have not yielded good results in reasonable amounts of computation time. Knowledge-based expert systems can solve these problems and yield good results in a reasonable amount of CPU time.

By combining expert systems with algorithmic approaches, we should be able to automate the entire custom chip design process. This development has profound implications. Automatically-produced designs can be guaranteed to be correct and can be produced far more quickly than if human intervention were required. The designer can either accept the automatic design or can use his knowledge to guide synthesis to a better result. Designers should be able to produce far better chip designs when aided by top-to-bottom synthesis aids than if they worked without CAD tools.

We still have much work to do, but we are encouraged by our results so far. We have demonstrated that the knowledge-based expert system methodology is an effective approach to high-level hardware synthesis. Our understanding of chip design leads us to believe that we can apply expert knowledge to the physical design of VLSI chips with similar success. □

Acknowledgments

The work described here has built on work done by us and others at Carnegie-Mellon University and Stanford University. At Carnegie-Mellon we would particularly like to thank S. Director, E. Dirkes, C. Forgy, D. Gatenby, C. Hitchcock, J. Lertola, J. McDermott, M. McFarland, J. Nestor, A. Newell, J. Rajan, D. Siewiorek, D. Thomas, and R. Walker for many ideas, probing questions, and software support.

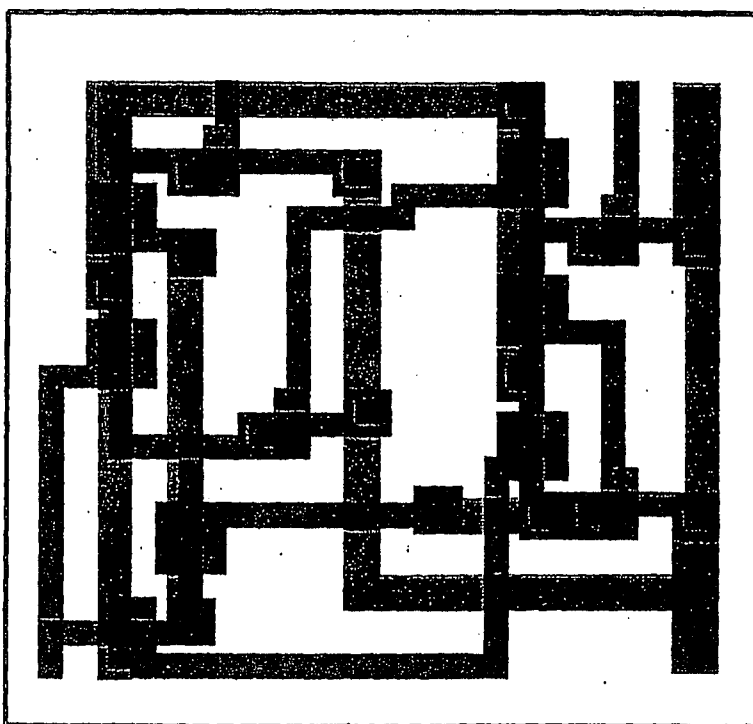


Figure 8. Layout in XYMASK form.

At Stanford we owe particular thanks to P. Eichenberger, D. Perkins, and T. Saxe, who designed and implemented Lava, and to R. Mathews, J. Newkirk, R. Dutton, and J. Hennessy for valuable discussions and support.

We would also like to thank J. Boddie, K. Chong, C. Davis, D. Ditzel, M. Maul, G. Mowery, A. Ross, C. Schneider, G. Williams, and A. Wilson for donating time to critique the designs. And we would like to thank J. Hartung, D. Hill, D. McCreary, and H. Moscovitz for taking time to discuss the problems of chip synthesis.

References

1. B.T. Murphy et al., "A CMOS 32b Single Chip Microprocessor," *Int'l Solid-State Circuits Conf.*, 1981, pp. 230-231.
2. R.N. Kershaw et al., "A Programmable Digital Signal Processor with 32b Floating Point Arithmetic," *32nd ISSCC*, Feb. 1985.
3. T.J. Kowalski and D.E. Thomas, "The VLSI Design Automation Assistant: An IBM System/370 Design," *IEEE Design and Test of Computers*, Vol. 1, No. 1, Feb. 1984, pp. 60-69.
4. R. Mathews, J. Newkirk, and P. Eichenberger, "A Target Language for Silicon Compilers," *Proc. Compcon*, Spring 1982, pp. 349-353.
5. N. Weste, "Virtual Grid Symbolic Layout," *18th Design Automation Conf.*, 1981, pp. 225-233.
6. W. Wolf, *Two-Dimensional Compaction Strategies*, PhD thesis, Stanford University, Stanford, Calif., Mar. 1984.
7. W. Wolf et al., "Dumbo, A Schematic-to-Layout Compiler," *3rd Caltech Conf. VLSI*, Mar. 1983, pp. 379-394.
8. T.J. Kowalski, *The VLSI Design Automation Assistant: A Knowledge-Based Expert System*, PhD thesis, Dept. Electrical and Computer Eng., Carnegie-Mellon University, Pittsburgh, Pa., Apr. 1984.
9. M.R. Barbacci et al., *The ISPS Computer Description Language*, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., Aug. 1979.
10. E. Dirkes, *A Module Binder for the CMU-DA System*, masters thesis, Dept. Electrical and Computer Eng., Carnegie-Mellon University, Pittsburgh, Pa., Apr. 1985.
11. D.J. Geiger, *A Framework for the Automatic Design of Controllers*, masters thesis, Research Report No. CMUCAD-84-34, Carnegie-Mellon University, Pittsburgh, Pa., July 1984.
12. R. Di Russo, *A Design Implementation Using the CMU-DA System*, Masters thesis, Dept. Electrical Eng., Carnegie-Mellon University, Pittsburgh, Pa., Oct. 1981.
13. L. Hafer, *Data-Memory Allocation in the Distributed Logic Design Style*, masters thesis, Carnegie-Mellon University, Pittsburgh, Pa., Dec. 1977.
14. A.C. Parker et al., "The CMU Design Automation System: An Example of Automated Data Path Design," *Proc. 16th Design Automation Conf.*, June 1979, pp. 73-80.
15. P. Marwedel and G. Zimmermann, *MIMOLA Software System User Manual*, 1, Institute Fur Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, May 1979.
16. L.J. Hafer, *Automated Data-Memory Synthesis: A Format Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic*, PhD thesis, Dept. Electrical Eng., Carnegie-Mellon University, Pittsburgh, Pa., June 1981.
17. C.J. Tseng and D.P. Siewiorek, "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proc. 20th Design Automation Conf.*, June 1983, pp. 490-496.
18. C.Y. Hitchcock, "Automated Synthesis of Data Paths," CMUCAD-83-4, SRC-CMU Center for CAD, Carnegie-Mellon University, Pittsburgh, Pa., Jan. 1983.
19. A.E. Dunlop, "Automatic Layout of Gate Arrays," *Proc. Int'l Symp. Circuits and Systems*, 1983, pp. 1245-1248.
20. B.T. Preas, *Placement and Routing Algorithms for Hierarchical Integrated Circuit Layout*, PhD thesis, Stanford University, Stanford, Calif., Aug. 1979.
21. C. Horng and M. Lie, "An Automatic/Interactive Layout Planning System for Arbitrarily Sized Rectangular Building Blocks," *Proc. 18th Design Automation Conf.*, 1981, pp. 293-300.
22. S.C. Johnson and S.A. Browning, *The LSI Design Language I*, AT&T Bell Labs internal memorandum, Murray Hill, N.J., 1980.
23. P.W. Kollaritsch and N.H.E. Weste, "A Rule-Based Symbolic Layout Expert," *VLSI Design*, Aug. 1981, pp. 62-66.
24. J. Kim and J. McDermott, "TALIB: An IC Layout Design Assistant," *Proc. AAAI*, 1983, pp. 197-201.
25. D. Johannsen, "Bristle Blocks: A Silicon Compiler," *Conf. VLSI*, Jan. 1979, pp. 303-310.
26. J.M. Siskind, J.R. Southard, and K.W. Crouch, "Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions," *Proc. Conf. Advanced Research in VLSI*, Jan. 1982, pp. 28-40.
27. K. Chu and S. Ramautar, "A Technology-Independent MOS Multiplier Generator," *Proc. 21st Design Automation Conf.*, 1984, pp. 90-97.
28. A. Newell, "Heuristic Programming: Ill-Structured Problems," in *Progress in Operations Research 3*, J. Aronofsky, ed., Wiley, New York, 1969, pp. 360-414.
29. J. Vasantharajan, *Design and Implementation of a VT-Based Multi-level Representation*, masters thesis, Dept. Electrical Eng., Carnegie-Mellon University, Pittsburgh, Pa., Feb. 1982.
30. G. De Micheli and A. Sangiovanni-Vincentelli, "KISS: A Program For Optimal State Assignment of Finite State Machines," *Int'l Conf. Computer-Aided Design*, Nov. 1984, pp. 209-211.
31. M.J. Meyer, P. Agrawal, and R.G. Phister, "A VLSI FSM Design System," *Proc. 21st Design Automation Conference*, June 1984, pp. 434-440.



Ted Kowalski is a member of technical staff in the Acoustical and Behavioral Research Center at AT&T Bell Laboratories. He is currently examining how VLSI designers choose a hardware architecture and whether a knowledge-based expert system can mimic their style. His research interests also include artificial intelligence, operating systems, programming and text-processing environments, and real-time systems.

Kowalski received the BSE in computer engineering from the University of Michigan in 1977, the MSEE in computer engineering from Carnegie-Mellon University in 1978, and a PhD in electrical engineering in 1984, also from Carnegie-Mellon University. He is a member of the Vulcans engineering honor-service society, Eta Kappa Nu, and Phi Beta Kappa.



Wayne Wolf was educated at Stanford University, receiving the BS in 1980, the MS in 1981, and the PhD in electrical engineering in 1984. He is currently a member of the technical staff at AT&T Bell Laboratories.

Wolf's current work is concentrated on the automatic design of layouts. His research interests include VLSI design methodologies, computer-aided design for VLSI, and novel machine architectures. He has been elected to Tau Beta Pi and Phi Beta Kappa, and is a member of IEEE and ACM.



Wolfgang Fichtner received the MS in physics and the PhD (cum laude) in electrical engineering, both from the Technische Universität, Vienna, Austria. He currently supervises a team researching VLSI device analysis and simulation.

Fichtner is a senior member of the IEEE, and a member of AAAI, ACM, APS, Electrochemical Society, and SIAM.



David Geiger is a graduate student at Carnegie-Mellon University. He is interested in using multilevel simulators to check the correctness of designs produced by the DAA system.

He received the BSEE in 1982 and the MSEE in 1984, both from Carnegie-Mellon University.

Authors Kowalski, Wolf, and Fichtner can be contacted at AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974; (201) 582-3000. Geiger can be contacted at Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE

Design & Test

of Computers

SP03997

There are three ways to subscribe... choose the best one for you.

☐ 1. Subscribe at the non-member rate of \$93/year.

☐ 2. Save by subscribing at the sister society rate. If you are a member of an affiliate society such as ACM, NCGA, ASQC, or any other professional technical society, you are eligible for a special rate of \$25/year. Fill in coupon below and mail with your remittance today.

☐ 3. Save even more by joining the IEEE Computer Society and subscribing to D&T at the low member rate of \$12/year (plus membership dues). Check box and we will send you a brochure that explains membership rates and benefits.

SAVE TIME

Call the Computer Society Membership Department at (714) 821-8380.

We will

- send you a membership brochure and application by return mail, or
- answer your questions; or
- send you a coupon good for a free book if you join the Society, or
- send you a free copy of *Computer Magazine*, or
- All of the above

Affiliate Society _____ Membership No. _____
(if applicable)

Name _____
(please type or print)

Street _____

City _____

State _____ Zip _____

Mo. | Yr.

☐ Check or money order enclosed. Exp. Date

☐ VISA ☐ MasterCard ☐ American Express

Charge Card Number

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Return to: IEEE Computer Society
10662 Los Vaqueros Circle • Los Alamitos, CA 90720

EXHIBIT

38

PTO/SB/57 (04-05)

Approved for use through 04/30/2007. OMB 0651-0033

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

(Also referred to as FORM PTO-1465)

REQUEST FOR EX PARTE REEXAMINATION TRANSMITTAL FORM

Address to:

Mail Stop Ex Parte Reexam
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

Attorney Docket No.: 8089.002

Date: 17 JANUARY 2006

1. ☒ This is a request for *ex parte* reexamination pursuant to 37 CFR 1.510 of patent number 4,922,432 issued May 1, 1990. The request is made by:
- ☐ patent owner. ☒ third party requester.
2. ☒ The name and address of the person requesting reexamination is:
- Novak Druce DeLuca & Quigg LLP
1300 Eye St., NW suite 400 East Tower
Washington DC 20005
3. ☒ a. A check in the amount of \$ _____ is enclosed to cover the reexamination fee, 37 CFR 1.20(c)(1);
- ☐ b. The Director is hereby authorized to charge the fee as set forth in 37 CFR 1.20(c)(1) to Deposit Account No. _____ (submit duplicative copy for fee processing); or
- ☒ c. Payment by credit card. Form PTO-2038 is attached.
4. ☒ Any refund should be made by ☐ check or ☒ credit to Deposit Account No. 14-1437. 37 CFR 1.26(c). If payment is made by credit card, refund must be to credit card account.
5. ☒ A copy of the patent to be reexamined having a double column format on one side of a separate paper is enclosed. 37 CFR 1.510(b)(4)
6. ☐ CD-ROM or CD-R in duplicate, Computer Program (Appendix) or large table
☐ Landscape Table on CD
7. ☐ Nucleotide and/or Amino Acid Sequence Submission
If applicable, items a. - c. are required.
- a. ☐ Computer Readable Form (CRF)
- b. Specification Sequence Listing on:
- i. ☐ CD-ROM (2 copies) or CD-R (2 copies); or
- ii. ☐ paper
- c. ☐ Statements verifying identity of above copies
8. ☒ A copy of any disclaimer, certificate of correction or reexamination certificate issued in the patent is included.
9. ☒ Reexamination of claim(s) 13 - 17 is requested.
10. ☒ A copy of every patent or printed publication relied upon is submitted herewith including a listing thereof on Form PTO/SB/08, PTO-1449, or equivalent.
11. ☐ An English language translation of all necessary and pertinent non-English language patents and/or printed publications is included.

[Page 1 of 2]

This collection of information is required by 37 CFR 1.510. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Mail Stop Ex Parte Reexam, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PTO/SB/57 (04-05)

Approved for use through 04/30/2007. OMB 0651-0033

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

12. <input checked="" type="checkbox"/> The attached detailed request includes at least the following items:		
a. A statement identifying each substantial new question of patentability based on prior patents and printed publications. 37 CFR 1.510(b)(1) b. An identification of every claim for which reexamination is requested, and a detailed explanation of the pertinency and manner of applying the cited art to every claim for which reexamination is requested. 37 CFR 1.510(b)(2)		
13. <input type="checkbox"/> A proposed amendment is included (only where the patent owner is the requester). 37 CFR 1.510(e)		
14. <input checked="" type="checkbox"/> a. It is certified that a copy of this request (if filed by other than the patent owner) has been served in its entirety on the patent owner as provided in 37 CFR 1.33(c). The name and address of the party served and the date of service are: <u>Ricoh Co., Ltd; Aeroflex, Inc.; AMI Semiconductor, Inc.;</u> <u>Matrox Electronic Systems, Ltd.</u> <u>(SEE ATTACHED SHEET FOR ADDRESSES)</u> Date of Service: <u>17 JAN 2006</u> ; or <input type="checkbox"/> b. A duplicate copy is enclosed since service on patent owner was not possible.		
15. Correspondence Address: Direct all communication about the reexamination to:		
<input checked="" type="checkbox"/> The address associated with Customer Number: 28694		
OR		
<input type="checkbox"/> Firm or Individual Name		
Address		
City	State	Zip
Country		
Telephone	Email	
16. <input checked="" type="checkbox"/> The patent is currently the subject of the following concurrent proceeding(s):		
<input type="checkbox"/> a. Copending reissue Application No. _____ <input type="checkbox"/> b. Copending reexamination Control No. _____ <input type="checkbox"/> c. Copending Interference No. _____ <input checked="" type="checkbox"/> d. Copending litigation styled:		
<u>Ricoh Co. Ltd. v. Aeroflex Inc. et al, No. 3:03cv4669 (ND Cal.)</u> <u>Synopsys, Inc. v. Ricoh Co. Ltd, No. 3:03cv 2289 (ND Cal.)</u>		
WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.		
<u>Vincent M De Luca</u> Authorized Signature		<u>17 JAN 2006</u> Date
<u>Vincent M De Luca</u> Typed/Printed Name		<u>32,408</u> Registration No.
		<input type="checkbox"/> For Patent Owner Requester <input checked="" type="checkbox"/> For Third Party Requester

Attachment

Ricoh Company, Ltd.

Altshuler Berzon Nussbaum Rubin &
Demain
177 Post Street, Suite 300
San Francisco, CA 94108

Aeroflex Incorporated

AMI Semiconductor, Inc.

Matrox Electronic Systems, Ltd.

HOWREY SIMON ARNOLD & WHITE, LLP
301 Ravenswood Avenue
Menlo Park, California 94025

Synopsys, Inc.

HOWREY SIMON ARNOLD & WHITE, LLP
301 Ravenswood Avenue
Menlo Park, California 94025

PTO/SB/088 (07-05)

Approved for use through 07/31/2006. OMB 0651-0031

U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Substitute for form 1449/PTO INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Use as many sheets as necessary)		Complete if Known	
		Application Number	Reexam of Pat. No. 4,922,432
		Filing Date	January 13, 1988
		First Named Inventor	Hideaki KOBAYASHI
		Art Unit	
		Examiner Name	
Sheet 1	of 1	Attorney Docket Number	8089.002

NON PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No. ¹	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T ²
		T.J. KOWALKSKI, D.J. Geiger, W. H. Wolf, W. Fichtner, The VLSI Design Automation Assistant: From Algorithms to Silicon, IEEE Design & Test, pp. 33-43 (1985)	
		Thaddeus Julius KOWALSKI, The VLSI Design Automation Assistant: A Knowledge-Based Expert System, Carnegie-Mellon University PhD Thesis, April 1984	

Examiner Signature		Date Considered	
--------------------	--	-----------------	--

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

¹ Applicant's unique citation designation number (optional). ² Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 (1-800-786-9199) and select option 2.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Reexamination of U.S. Patent No. 4,922,432

Named Inventor(s): Hideaki Kobayashi et al.

Issued: May 1, 1990

Filed: January 13, 1988

Title: KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING
INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

Assignee: Ricoh Co., Ltd.

REQUEST FOR REEXAMINATION

Mail Stop *Ex Parte* Reexam
Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Reexamination under 35 U.S.C. § 302-307 and 37 C.F.R. § 1.510 is respectfully requested of United States Patent No. 4,922,432 ("the '432 patent"), entitled "Knowledge Based Method and Apparatus for Designing Integrated Circuits Using Functional Specifications," which issued May 1, 1990 from Serial No. 143,821, filed January 13, 1988. A copy of the '432 patent, in the format specified by 37 C.F.R. § 1.510(b) (4), is included with this request. As claims 13-17 of the '432 patent are currently involved in ongoing patent infringement litigation, this request should be processed with "special dispatch" pursuant to 35 U.S.C. § 305, and be given "priority over all other cases" as required by MPEP § 2261.

I. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED AND CITATION OF PRIOR ART

Reexamination is requested of claims 13-17 of the '432 patent in view of the prior art listed below.

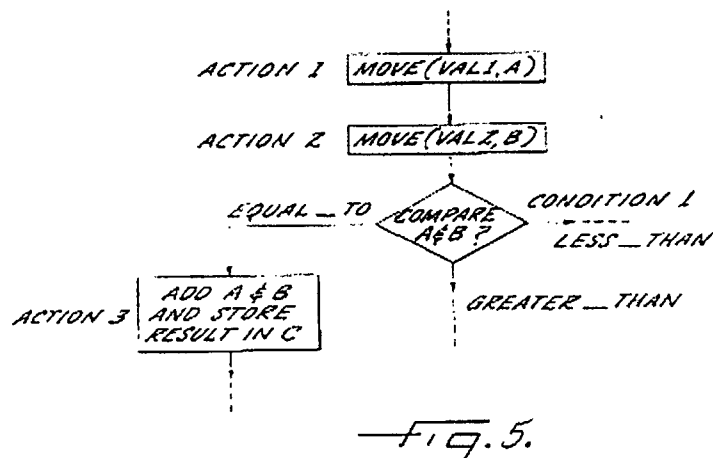
ITEM	PRIOR ART REFERENCE
1.	T.J. Kowalski, D.J. Geiger, W. H. Wolf, W. Fichtner, The VLSI Design Automation Assistant: From Algorithms to Silicon, IEEE Design & Test, pp. 33-43 (1985) ("Kowalski85")
2.	Thaddeus Julius Kowalski, The VLSI Design Automation Assistant: A Knowledge-Based Expert System, Carnegie-Mellon University PhD Thesis, April 1984 ("Kowalski84")

The above-identified prior art references are listed on accompanying form PTO-1449. Copies of each of these references are included with this Request.

II. ALLEGED INVENTION OF THE '432 PATENT

According to the specification of the '432 patent, the alleged invention "relates to a computer-aided method and apparatus for designing integrated circuits." Col. 1:10-12. The integrated circuits are intended to perform a specific function, and are known as application specific integrated circuits ("ASICs"). Col. 1:13-15. The method and apparatus of the '432 patent is directed towards allowing a designer, who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer, to design these ASIC chips aided by Computer Assisted Design ("CAD") technology. Col 2:6-20.

Thus, according to the '432 patent, the designer first specifies the functions to be performed by the intended chip at an architecture independent level. Col 2:21-24. The functional specification can be in the form of a flowchart. *Id.* Figure 5 of the '432 patent, reproduced below, illustrates an example of a flowchart description of the functions of the intended chip:



The boxes represent actions to be performed by the chip, while the diamonds represent conditions to be tested within the chip. Col 4:61-63; Col 7:20-23.

Using CAD techniques, the method and apparatus of the '432 patent then allows a user to associate the actions and conditions of the functional description with pre-defined definitions, generally in the form of a macro of the type illustrated in Table I of the '432 patent, Col. 7:24-26, reproduced below, drawn from a macro library, Col. 5:20-22:

TABLE 1

Macro	Description
ADD (A,B,C)	$C = A + B$
SUB (A,B,C)	$C = A - B$
MULT (A,B,C)	$C = A * B$
DIV (A,B,C)	$C = A \div B$
DECR (A)	$A = A - 1$
INCR (A)	$A = A + 1$
CLR (A)	$A = 0$
REG (A,B)	$B = A$
CMP (A,B)	Compare A to B and set EQ,LT,GT signals
CMPO (A)	Compare A to 0 and set EQ,LT,GT signals
NEGATE (A)	$A = \text{NOT}(A)$
MOD (A,B,C)	$C = A \text{ Modulus } B$
POW (A,B,C)	$C = A^B$
DC2 (A,S1,S2,S3,S4)	Decode A into S1,S2,S3,S4
EC3 (S1,S2,S3,S4,A)	Encode S1,S2,S3,S4 into A
MOVE (A,B)	$B = A$
CALL (sub-flowchart (A,B,...))	Call a sub-flowchart. Pass A,B,...
START (A,B,...)	Beginning state of a sub-flowchart
STOP (A,B,...)	Ending state of a sub-flowchart

For example, the specified action “ADD A & B AND STORE RESULT IN C” from the flowchart of Figure 5 corresponds to the macro ADD (A,B,C) from the above table.

A Knowledge Based Silicon Compiler (“KBSC”), a computer program described in the specification of the ‘432 patent, includes a “path synthesizer and cell selector” (“PCSC”) that allocates technology dependent hardware “cells” or modules, such as registers, comparators and the like, to implement the macro definitions. Col. 5:14-30. A control generator within the KBSC also generates the control logic to control the functioning of these cells or modules, as well as the paths linking these modules together. Col. 5:3-11. Figure 6 of the ‘432 patent, reproduced below, shows the hardware cells, paths and control logic resulting from applying the KBSC to the macro definitions corresponding to the flowchart description of Figure 5:

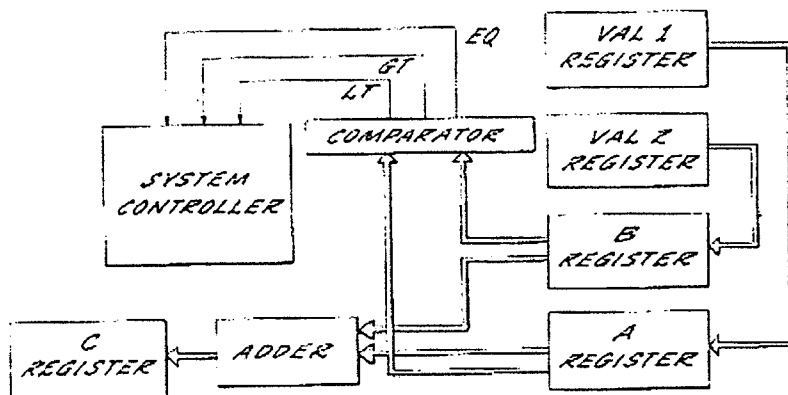


FIG. 6.

In a preferred embodiment, illustrated in Figure 3, reproduced below, the PSCS 30 uses a knowledge base 35, representing knowledge extracted from expert ASIC designers, Col. 2:58-61; Col. 8:29-30, to map the macro definitions of the architecture independent actions and conditions into the technology dependent cells. Col. 8:34-37; Col. 8:58-64; Col. 9:21-22. PSCS 30 also allocates a system controller and linking data and control paths. Col. 5:8-11.

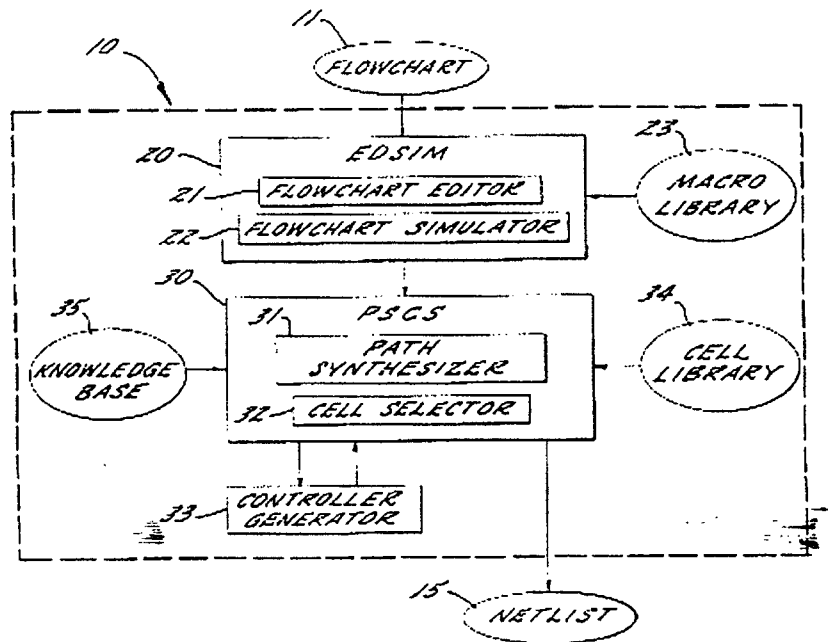


FIG. 3.

The expert knowledge in the knowledge base 35 is preferably in the form of "IF-THEN-ELSE" formatted rules such as the following:

- 7 -

According to the '432 specification, the cells and control logic allocated by the PCSC, together with the data and controls paths linking these elements together, form a "netlist," i.e., a list that identifies each allocated block in the resulting circuit, and the interconnections between the respective inputs and outputs of the blocks. Col. 5:36-38.

Additionally, according to the '432 specification, commercially available CAD systems receive the "netlist" as an input, and produce there-from "mask" data describing the layout of the respective cells in the chip as well as the routing between the cells. Col. 5:40-46. Figure 2 of the '432 patent, reproduced below, illustrates this overall process:

PCSC Example Rules	
Rule 1	IF no blocks exist
Rule 2	THEN generate a system controller
IF	a state exists which has a macro AND this macro has not been mapped in a block find a corresponding macro in the library
THEN	and generate a block for this macro
Rule 3	IF there is a transition between two states AND there are macros in these states using the same argument make a connection from a register corresponding to the first macro to another register corresponding to the second macro
THEN	a register has only a single connection from another register combine these registers into a single register.
Rule 4	IF there are two comparators AND input data widths are of the same size AND
THEN	
Rule 5	IF

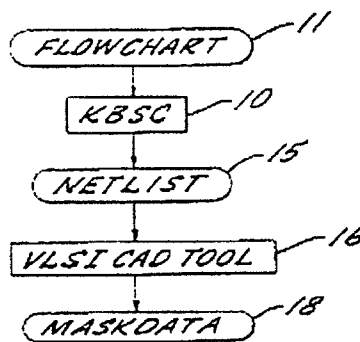


FIG. 2.

III. PENDING LITIGATION

In *Ricoh Co., Ltd. v. Aeroflex, Inc., et al.*, No. C 03-4669, an ongoing patent infringement action pending in the United States District Court for the Northern District of California, Ricoh Co., Ltd. ("Ricoh"), the current patent owner, has asserted that various entities infringe claims 13-17 of the '432 patent. In *Synopsys, Inc. v. Ricoh Co., Ltd.*, No. C 03-2289, an ongoing declaratory judgment action also pending in the United States District Court for the Northern District of California, Synopsys, Inc. has requested a declaratory judgment that Synopsys' products do not infringe claims 13-17 of the '432 patent and that the patent is invalid. Due to this ongoing litigation, as stated at the outset, this request should be processed with "special dispatch" pursuant to 35 U.S.C. §305, and given "priority over all other cases" as required by MPEP §2261.¹ Additionally, should a reexamination be ordered, the Patent Office is respectfully requested to indicate the expected timing of a first Office Action.

IV. THE LEGAL STANDARD GOVERNING THIS REEXAMINATION

During reexamination, the Patent Office, under controlling authority, is obligated to give each claim term its broadest reasonable interpretation consistent with the specification. More specifically, 37 CFR § 1.555(b)(2)(ii), states in pertinent part:

¹ Copies of the complaints in these two actions are attached pursuant to MPEP 2219.

A *prima facie* case of unpatentability of a claim pending in a reexamination proceeding is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability. (underlining added; italics in original).

Moreover, the Federal Circuit has specifically held that, during reexamination, “claims . . . are to be given their broadest reasonable interpretation consistent with the specification, and . . . claim language should be read in light of the specification as it would be interpreted by one of ordinary skill in the art.” *In re American Academy of Science Tech. Center*, 367 F.3d 1359, 1364 (Fed. Cir. 2004) (quoting ~~in~~ *re Bond*, 910 F.2d 831, 833 (Fed. Cir. 1990)); ~~see also~~ *In re Yamamoto*, 740 F.2d 1569, 1571 (Fed. Cir. 1984).

Such a procedure “serves the public interest by reducing the possibility that claims, finally allowed, will be given broader scope than is justified.” *Yamamoto*, 740 F.2d at 1571; *see also In re Hyatt*, 211 F.3d 1367, 1372 (Fed. Cir. 2000); *In re Zletz*, 893 F.2d 319, 322 (Fed. Cir. 1989). It also “promotes the development of the written record before the PTO that provides the requisite written notice to the public as to what the applicant claims as the invention.” *In re Morris*, 127 F.3d 1048, 1054 (Fed. Cir. 1997). This procedure is fair to the patentee, who always has the opportunity to amend the claims during reexamination. *See Yamamoto*, 740 F.2d at 1571-72.

In the co-pending litigations referred to in the previous section, the district court has entered a claim construction order after claim construction briefing by the parties. A copy of the district court’s claim construction order, as well as Ricoh’s opening and reply claim construction briefs, are included with this request.

Because the Patent Office is obligated to give each claim term its broadest reasonable interpretation, it is not bound to the construction set forth in the district court’s claim construction order. Nonetheless, Ricoh’s proposed construction of the claim terms, as set forth in its claim construction briefing, is relevant as it constitutes admissions by Ricoh as to the

broadest reasonable interpretation that is supported by the specification. The district court's claim construction order may also be relevant, although, because it is not a final judgment and also non-binding on the Patent Office, it is less pertinent than the admissions in Ricoh's claim construction briefing. Therefore, the Patent Office, in formulating this broadest reasonable interpretation, should primarily consider Ricoh's assertions as to the alleged breadth of the claims.

Consider, for example, the term "rules" in element [3] of claim 13, *i.e.*, the step of "storing in an expert system a set of *rules* for selecting hardware cells to perform the actions and conditions." In its claim construction reply brief, Ricoh disagreed that the "expert knowledge" embodied in these "rules" was limited to "'mapping the specified definitions in the flowchart to the hardware cell descriptions.'" See Ricoh Claim Construction Reply Br., at p. 11. Rather, Ricoh argued, "nothing in the language of claim 13 itself (or anything in the public record) requires such a limitation on the "rules" of claim 13." See Ricoh Claim Construction Reply Br., at p. 12. In its claim construction order, the district court accepted Ricoh's position that these "rules" need not "encompass the 'mapping' function." See Claim Construction Order, p. 18.

Another example is element [6] of claim 13, *i.e.*, the step of selecting hardware cells.² In its claim construction reply brief, Ricoh disagreed that this step "must be performed by a rule-based expert system and not conventional software." See Ricoh Claim Construction Reply Br., at p. 15. Instead, Ricoh urged, "[n]othing . . . in the claim language dictates any means for applying the claimed rules." *Id.* While the district court, in its claim construction order, did not fully accept Ricoh's position, it did accept Ricoh's position that this element does not require

² Element [6] of claim 13 recites:

"selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefore."

an expert system in the traditional sense, i.e., as requiring an inference engine. See Claim Construction Order, pp. 15-16.

Another example is element [1], “storing a set of definitions of architecture independent actions and conditions.” During claim construction, Ricoh asserted the “architecture independent actions and conditions” were “functional or behavioral aspects of a portion of a circuit (or circuit segment) that does not imply any set architecture, structure or implementing technology.” See Ricoh Opening Claim Construction Brief, at p. 14.

Because of the different standards of claim interpretation that must be used by the courts and the Office, statements made in this Reexamination Request for purposes of establishing the existence of substantial new questions of patentability, are not admissible and cannot be used in a district court proceeding for any purpose.

**V. EXPLANATION OF PERTINENCY AND APPLICATION OF
CITED PRIOR ART TO EVERY CLAIM FOR WHICH
REEXAMINATION IS REQUESTED**

**A. This Art Is More Pertinent To The Claimed '432 Invention
Than The References Applied During The Original
Prosecution**

Claims 20-26 in Serial No. 143,821, filed January 13, 1988, issued as claims 13-17 in the '432 patent (after renumbering, cancellation and amendment). In the first Office Action, dated January 18, 1989, the Examiner rejected these claims as obvious based on U.S. Patent No. 4,703,435, Darringer et al., in view of J. H. Nash, A Front End Graphic Interface to the First Silicon Compiler, European Conference On Electronic Design Automation (EDA 84), pp. 123-124, March 1984. See Paper No. 3, pp. 2-3.³ In response, in an Amendment dated April 18, 1989, the applicant amended then claim 20 (now claim 13) to add the term “architecture independent” to elements [1] and [3]. See Paper No. 6, p. 6. Since that term did not appear in the specification, the applicant contemporaneously amended the specification to add it. *Id.* at pp.

³ A copy of this paper, and each of the other papers from the '432 prosecution history referred to herein, are attached to this Request.

1-2. In the accompanying remarks, applicant asserted that Darringer does not meet this newly minted “architecture independent” limitation:

Although Darringer et al. does disclose a method and system for automatic logic design and it is known in the art of automatic layout to utilize cell libraries of circuit components, Darringer does not teach the present invention. A very clear distinction between Darringer and the present invention is that the input to the Darringer system is in the form of a register transfer level flowchart control language.

* * *

In contrast, the application specific circuit designer utilizing the present invention need not possess any expertise common among highly skilled VLSI design engineers since input to the present invention is in the form of an architecture independent functional specification.

Id. at p. 9. Additionally, applicant argued, Darringer lacks an expert system knowledge base for generating, *i.e.*, synthesizing, the ASIC design. *Id.* at pp. 9-10. Applicant argued that Nash was distinguishable on these same bases. *Id.* at pp. 11-12.

Nonetheless, in the next Office Action, dated August 15, 1989, the Examiner adhered to his position by finally rejecting the claims as obvious over Darringer in view of Nash. *See* Paper No. 7, at pp. 2-3. In response, on October 19, 1989, the applicant conducted a personal interview with the Examiner to discuss the Darringer reference. *See* Paper No. 8. Claim 20 (now claim 13) was specifically discussed. *Id.* During the interview, the applicant argued that two features—a “flowchart editor” and “expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit”—patentably distinguished over Darringer. *Id.* The applicant agreed to “amend the claims to include these features.” *Id.*

However, despite these assurances, the applicant did not further amend claim 20 to expressly add these two features.⁴ Instead, after canceling then claims 21 and 25, applicant

⁴ The Examiner’s attention is particularly directed to the record on the issue of the failure of Patent Owner to explicitly amend claims 13-17 to include the identified “flowchart editor” and “expert system” limitations – which the applicant clearly represented were the alleged patentable distinctions over the prior art. Claims 13-17 appear to have been allowed without these explicit limitations despite the Patent Owner’s argument that such limitations were what made the invention patentable over the prior art, and the apparent meeting of the minds between the Examiner and the Patent Owner that these limitations would be added to claims 13-17.

merely amended then claim 20 (now claim 13) to incorporate an “expert system knowledge base” feature, *see* Paper No. 9, pp. 4-5, a feature that Ricoh now asserts is susceptible to a broader reasonable interpretation than the “expert system” limitation discussed at the interview. *See* Claim Construction Order, pp. 14-17. Nor did applicant ever expressly add a “flowchart editor” limitation to then claim 20. Indeed, Ricoh now asserts, and the district court has so ruled, that claim 13 does not incorporate a “flowchart editor” limitation. *See* Ricoh Opening Claim Construction Br., at pp. 17-18; Claim Construction Order, pp. 8-12.

Notwithstanding the foregoing, in accompanying remarks, applicant continued to assert that the “architecture independent” and “expert system” features were the defining features of the claimed invention that distinguished over the prior art:

[T]he present invention distinguishes fundamentally over the prior art by providing a system and method for designing an application specific integrated circuit at an architecture independent functional behavioral level. . . . The invention makes use of artificial intelligence technology, i.e., an expert system, to translate the architecture independent functional level specifications into an architecture specific structural level definition which can be used directly to produce an application specific integrated circuit. It was noted that while the Darringer et al. patent refers at some points in the specification to a so called “functional description”, it is clear from a complete reading of the patent specification in context that the specifications used by Darringer et al. are not truly at an architecture independent level, but rather are at a lower level which is indeed hardware architecture dependent and defines the system at a “register-transfer” level description. . . .

During the interview, the Examiner carefully reconsidered the prior art and applicants’ claims, and upon reconsideration agreed that certain features as defined in applicants’ claims, such as the “flowchart editor” and the “expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit” patentably distinguish applicants’ invention from the prior art of record, including Darringer et al., 4,703, 435.

Id. at pp. 6-7. In response, the Examiner allowed the claims.

In light of the foregoing, the applicant plainly considered the “architecture independent” and “expert system” features to be the defining features of the claimed invention that were unmet by the

prior art.⁵ However, as discussed in the Section D, *infra*, the prior art included with this request does meet these two features, particularly under the broadest reasonable interpretation of the claim language that applies to this reexamination. Therefore, it is more pertinent than the prior art applied during the original prosecution.

B. The Named Inventors Admitted the Materiality of the Art Included With this Request in a Contemporaneous Publication

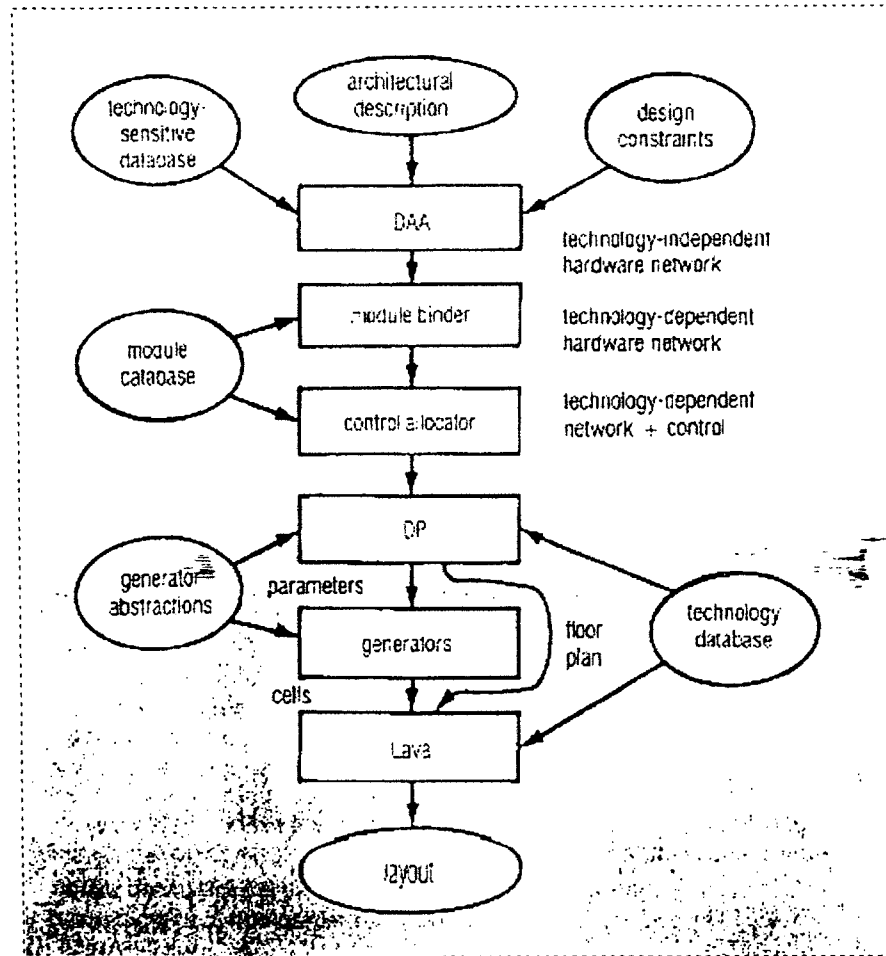
In 1989, contemporaneously with the original prosecution, the two named inventors of the '432 patent published a paper describing the very same KBSC system that is the subject of the '432 patent: Kobayashi J. et al., KBSC: A Knowledge-Based Approach to Automatic Logic Synthesis, International Journal of Computer Aided ALSI Design I, pp. 377-390 (1989) (“Kobayashi89”), a copy of which is attached. Kobayashi89 includes a literature survey of prior art logic synthesis systems. Of note, Kowalski85, the prior art that forms the basis for this re-examination request, is identified at p. 390 of Kobayashi89 as reference [5], demonstrating that the named inventors knew of this art during the pendency of the original prosecution. They also knew of the materiality of this art. Significantly, while claiming “KBSC is clearly distinguished from other logic synthesis systems in terms of its flowchart input form and rule-based approach to automatic data-path and control logic synthesis,” *see* Kobayashi89, at p. 389, they specifically admitted “the Design Automation Assistant (DAA) [of Kowalski85] is an expert system that uses heuristic rules to synthesize architectural implementation from an algorithmic description with design constraints.” *Id.* In other words, the named inventors admitted contemporaneously with the original prosecution that the DAA was a rules-based, expert system approach, and that the KBSC only distinguished from the DAA on the basis of a single feature, a flowchart editor in the KBSC, which the named inventors aver, is not present in claims 13-17. Despite this knowledge and materiality, however, applicant did not disclose Kowalski85 to the Patent Office.

⁵ Although the Examiner also found the “flowchart editor” feature to be a defining feature of the claimed invention, Ricoh has asserted, and the district court has so ruled, that claims 13-17 are not limited to this feature. *See* Ricoh Opening Claim Construction Br., at pp. 17-18; Claim Construction Order, pp. 8-12.

C. Overview of the New Art

Kowalski85 features a Design Automation Assistant (“DAA”) for synthesizing ASIC designs. *Id.* at p. 35. Kowalski85 refers to the DAA as functioning software. *Id.* (“[t]he DAA has been used to design several digital systems”). At page 34, in introducing the DAA, Kowalski85 refers to Kowalski84, and at page 42, lists Kowalski84 as reference [8] in the list of references. Consequently, one of skill in the art, when reviewing Kowalski85, would understand the teachings of Kowalski84 regarding the DAA to be inherent within Kowalski85. As such, the teachings of Kowalski84 regarding the DAA are part of the anticipating teachings of Kowalski85 as they merely explain that which is inherent in Kowalski85. *See Telemac Cellular Corp. v. Topp Telecom, Inc.*, 247 F.3d 1316, 1328 (Fed. Cir. 2001) (recourse to extrinsic evidence appropriate to determine that which is inherent in the disclosure of an anticipating reference). As both Kowalski85 and Kowalski84 were published prior to the critical date of January 13, 1987 (*i.e.*, more than one year prior to the filing date of the '432 patent), both are printed publications that anticipate claims 13-17 under 35 U.S.C. §102(b).

Like the '432 patent, Kowalski85 discloses a multi-stage, CAD-assisted process for designing a special-purpose VLSI chip. *Id.* at p. 34. Figure 1 of Kowalski85, reproduced below, illustrates this process:



In Kowalski85, as in the '432 patent, a user interface is provided that allows the designer to input an algorithmic description of the desired functionality of the chip, *id.* at p. 34:

We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The

The algorithmic description is written by the chip designer in the ISPS (Instruction Set Processor Specification) language, *see* Kowalski85, p. 35. In ISPS, a user specifies actions and

conditions for the desired chip as a functional algorithmic description. See Kowalski84, pp. 46-47.

A compiler, a program stored and executing on a computer, receives an ISPS source file and produces there-from a dataflow graph representation. See Kowalski85, p. 36. In the process of producing the dataflow representation, the compiler translates each of the actions and conditions into an operator—that is predefined—which forms a node of a graph. See Kowalski84, p. 49:

The ISPS description is compiled into a VT^{17,18} data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called *operators* and correspond to operations that take certain values as input and produce new values as output. They are translations of the ISPS unary and binary operations, operations that change or access fields in words or words in arrays, control operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called *ownodes* and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called *VT-bodies*, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops.

The DAA then selects, for each of the operators (corresponding to the specified architecture independent actions and conditions) at the nodes of the graph, one or more technology sensitive modules, see Kowalski85, p. 35:

ISPS,⁹ the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs.

More specifically, the DAA produces a technology sensitive hardware network composed of modules, ports, links, and symbolic microcode, *see* Kowalski85, p. 36:

The DAA produces a technology-independent hardware network. This network is composed of modules, ports, links, and symbolic microcode. The modules can be registers, operators, memories, and buses or MUXes with input, output, and bidirectional ports. The ports are connected by links and are controlled by the symbolic microcode.

In doing the selecting, the DAA applies a set of rules, each embodying the knowledge of expert VLSI chip designers and stored in an expert system knowledge base, to the graph, *see* Kowalski85, p. 36:

DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize an architectural implementation from an algorithmic description with constraints. The de-

The rules in the DAA are described in detail in Chapter 5 of Kowalski84, and the types of rules summarized in Table 14 of Kowalski84, p. 63, as follows:

Table 14. RULES BY FUNCTION AND KNOWLEDGE TYPE

Section	Rules	Firings	Design	Context	Setup	Cleanup	Input	List	Test
Total DAA	314	8543	151	18	70	33	9	14	19
Service Functions	88	4901	70	7	0	3	0	4	4
Control management	12	1981	0	7	0	1	0	4	0
Module management	21	722	19	0	0	2	0	0	0
Peri management	34	732	32	0	0	0	0	0	2
Link management	21	1466	19	0	0	0	0	0	2
Global Allocation	78	104	7	0	70	1	0	0	0
Memories	4	13	3	0	0	1	0	0	0
Registers	2	10	2	0	0	0	0	0	0
Constants	1	10	1	0	0	0	0	0	0
Controller	1	1	1	0	0	0	0	0	0
Technology database	70	70	0	0	70	0	0	0	0
Value Trace Allocation	55	1468	25	0	0	13	9	6	2
Initialization	(See Cleanup)								
Operator assignment	9	322	4	0	0	3	0	2	0
Operator trimming	13	64	0	0	0	2	9	0	2
Temporary registers	2	107	1	0	0	1	0	0	0
Register functions	6	18	6	0	0	0	0	0	0
ALU functions	25	957	14	0	0	7	0	4	0
SCS Allocation	47	1610	21	6	0	9	0	0	11
Register stability	2	173	4	0	0	1	0	0	3
Register allocating	9	26	4	2	0	0	0	0	2
Module stability	5	6	5	0	0	0	0	0	0
Module allocating	18	93	8	4	0	0	0	0	6
Cleanup	8	1312	0	0	0	8	0	0	0
Global Improvements	46	460	28	5	0	7	0	4	2
Unreferenced	6	58	5	0	0	1	0	0	0
Multiplexer cleanup	7	14	5	0	0	1	0	1	0
Multiple fan-out cleanup	4	12	0	3	0	1	0	0	0
Bus utilization	29	376	18	2	0	4	0	3	2
Final cleanup	(See Unreferenced and Multiplexer cleanup)								

Each of the rules is written in an antecedent IF-THEN-ELSE format, see Kowalski84, p.

28:

3.3.2 The rule memory. The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:

IF:
 the most current active context is to create a link
 and the link should go from a source port to a destination port
 and the module of the source port is not a multiplexer
 and there is a link from another module to the same destination port
 and this other module is not a multiplexer
THEN:
 create a multiplexer module
 and connect the multiplexer to the destination port
 and connect the source port and destination port link to the multiplexer
 and move the other link from the destination port to the multiplexer

This rule recognizes situations in which a multiplexer needs to be created to connect one port to another.

Each subtask in the DAA is associated with a set of rules for carrying out the subtask. An example of a rule for the fourth subtask appeared above. Most of the rules, like the example above, define situations in which a partial design should be extended in some particular way. These rules enable the DAA to synthesize an acceptable design by determining, at each step, whether a certain design extension respects constraints.

A module binder then binds an integrated circuit hardware module or "cell" to each of the technology sensitive modules produced by the DAA, see Kowalski85, p. 37:

Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint.

The module binder either chooses a cell from a module database based on user-supplied constraints, or fabricates it from other database entries, *see* Kowalski85, p. 35:

• The module binder¹⁰: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either selected from the module database or fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University.

The result of the module binder is a technology-dependent hardware network, *e.g.*, a netlist. *See* Kowalski85, Figure 1, p. 34.

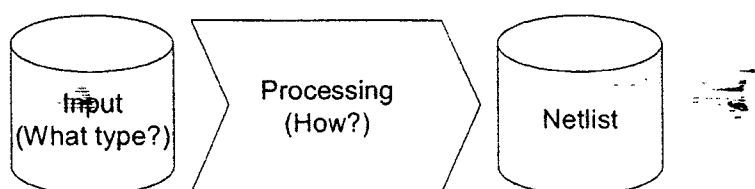
D. Application of this New Prior Art to Claims 13-17

Claim 13 consists of a preamble and six elements in the body, which can be referred to as elements [1] – [6]. Claims 14 and 15 depend from claim 13, claim 16 depends from claim 15, and claim 17 depends from claim 16. These claims are set forth in their entirety in the claim charts that are presented below. Before presenting these claim charts, it is helpful to highlight the following two central issues surrounding claim 13 that were the focus of the prosecution of

the '432 patent and the contemporaneous article of the named inventors (*see* Kobayashi89 and discussion, above):

- (1) What type of input to the system is required? (primarily relevant to elements [1], [2], [4] and [5] of claim 13)
- (2) How does the system process the input? (primarily relevant to elements [3] and [6] of claim 13)

The following figure graphically illustrates these two issues:



Regarding the first issue, the broadest reasonable interpretation of the claim language requires that the input must be “architecture independent,” without limitation to whether it is produced from a “flowchart editor.” Indeed, Ricoh now asserts, and the district court has so ruled, that claim 13 does not incorporate a “flowchart editor” limitation. *See* Ricoh Opening Claim Construction Br., at pp. 17-18; Claim Construction Order, pp. 8-12.

Applying Ricoh’s proposed construction as the broadest reasonable construction, Kowalski85 clearly teaches an “architecture independent” input,” *see* Kowalski85 at p. 34:

We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip’s functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS.

Notably, the above quote from page 34 of Kowalski85 closely tracks Ricoh's proposed construction of "architecture independent" in its claim construction briefing, namely, "functional or behavioral aspects of a portion of a circuit (or circuit segment) that does not imply any set architecture, structure or implementing technology." See Ricoh Opening Claim Construction Br., at p. 14. And the algorithmic input to Kowalski85 does not specify any set architecture, structure or implementing technology.⁶ Thus, element [1] is met. For similar reasons, as demonstrated in the claim charts below, elements [2], [4] and [5] of claim 13 are also met.

The second issue, which addresses the "rules," as well as their functionality and format, divides into the following two sub-issues:

- (2A) What functions must the rules perform, *i.e.*, mapping (*e.g.* hardware cell selection) or something broader?
- (2B) What format must the rules be in, *i.e.*, must the rules be contained in an expert system or can they be embodied as conventional algorithmic software?

Requestor's position is that Kowalski85 meets elements [3] and [6] of claim 13, irrespective of the answers to (2A) and (2B), thus rendering the entirety of claim 13 anticipated.

Regarding sub-issue (2A), Patent Owner has forcefully argued that the broadest reasonable interpretation of "rules" in element [3] of claim 13 is not limited to selecting hardware cells, referred to by Requestor as mapping:

24	more than just expert knowledge of highly skilled VLSI designers. Defendants argue that the "expert
25	knowledge" be specifically directed to "mapping the specified definitions in the flowchart to the hardware
26	cell descriptions." <i>Id.</i> Nothing in the language of claims 1, 9, or 18, nor the '432 patent specification
1	passage cited by Defendants, refers to "expert knowledge" for "mapping the specified definitions in the
2	flowchart to the hardware cell descriptions," as contended by Defendants. Even if these sources had
3	recited the use of such "expert knowledge," nothing in the language of claim 13 itself (or anything else in
4	the public record) requires such a limitation on the "rules" of claim 13.

⁶ It is also at Nor does Kowalski85 use a higher register transfer level of abstraction than description as taught in Darringer-RTL, and thus therefore, also meets this limitation applying the district court's Court's construction. See Claim Construction Order, at p. 12.

See Ricoh's Claim Construction Reply Brief, at p. 11:24-26 and p. 12:1-4. The district court did not explicitly construe the term "rules" in element [3] of claim 13. However, in construing the similar phrase "a set of cell selection rules" in element [6], the court stated that it was adopting Ricoh's position that the plain language of "set of cell selection rules" (element [6] of claim 13) "does not dictate that the 'rules' encompass the 'mapping function'." See Claim Construction Order, at p. 18. Accordingly, Ricoh should be held to that position now for purposes of this reexamination, and the broadest reasonable interpretation of "rules" in elements [3] and [6] held to include rules for more than cell selection mapping.

Turning to issue (2B), Patent Owner forcefully argued that the broadest reasonable interpretation of "expert system knowledge base" in elements [3] and [6] is not limited to expert systems in the traditional sense, *i.e.*, as including an inference engine, but encompasses conventional algorithmic software:

1 the claims, the patent specification, or the file history.¹⁴ Thus, despite the fact that the plain language of
 2 claim 13 does not limit the means or mechanism that may be used to perform the claimed process, without
 3 any authority, Defendants would have this Court not only impose a limitation as to what type of means
 4 (i.e., an expert system) can be used, but also what type of means cannot be used (i.e., an algorithmic
 5 software). Relying solely on extrinsic evidence, Defendants allege (at 44) that "[a] person of ordinary skill
 6 in the art in 1988 would have known that the rule-based expert system software approach is substantially
 7 different than using conventional algorithmic software." Because such extrinsic evidence should not be
 8 relied upon, Defendants' unique proposal should be rejected on this basis alone. It should also be rejected
 9 because it fails to give the claims their proper scope based on their plain and ordinary meaning, and
 10 restricting the scope in the absence of any manifest exclusion or clear disavowal. *Tex. Digital Sys.*, 308
 11 F.3d at 1204.

See Ricoh's Reply Claim Construction Brief, page 17:1-11. The district court adopted Ricoh's position. See Claim Construction Order at pp. 14-17. Accordingly, Ricoh should be held to that position now for purposes of this reexamination, and the broadest reasonable interpretation of "rules" in elements [3] and [6] held to include rules embodied as conventional algorithmic software.

Under these interpretations, the rules in the DAA, which embody the expert knowledge of VLSI designers and are responsible for mapping the VT dataflow representation of the ISPS architecture independent description into technology sensitive modules, *see* Kowalski85, at p. 35, meet elements [3] and [6] of claim 13. Even if the rules under these interpretations are held to require some sort of mapping or selecting of hardware cells, the technology sensitive modules produced by the DAA, *see* Kowalski85, at p. 34, are “hardware cells” under the broadest reasonable interpretation of that term.

In particular, at Col. 6:28-29, the '432 patent characterizes the blocks shown below in Fig. 6 as “hardware cells”:

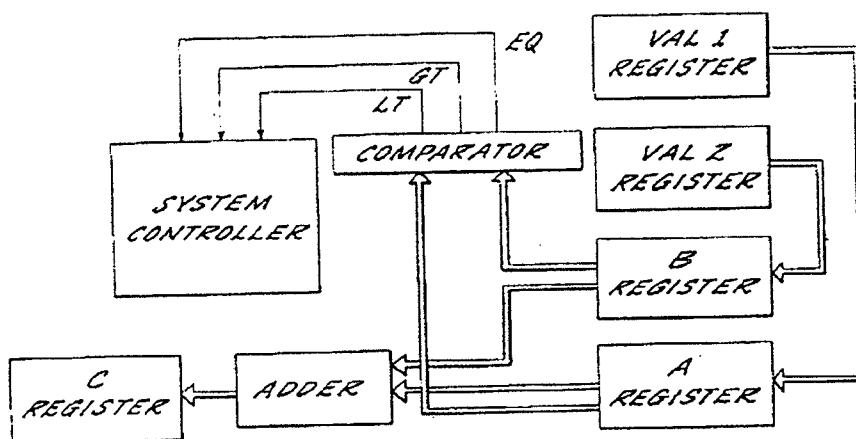


FIG. 6.

Certainly, the technology sensitive modules produced by the DAA, an example of which is illustrated below in Fig. 3 from Kowalski85, p. 37, represent a similar level of technology dependency and accordingly are “hardware cells” under the broadest reasonable interpretation:

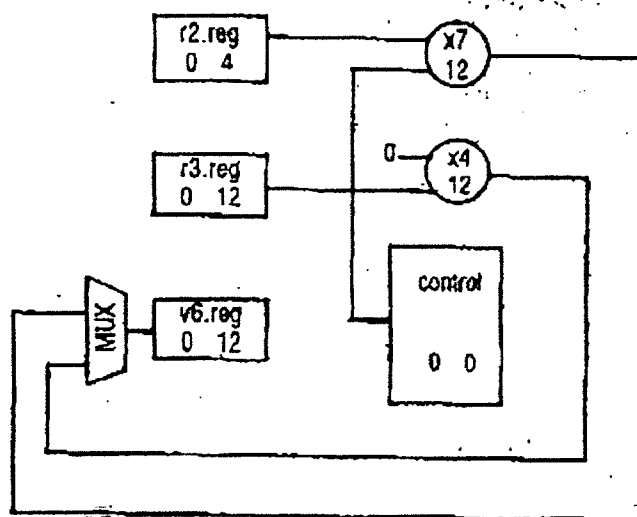


Figure 3. Sample structural specification.

Moreover, in the district court litigation, Ricoh urged the following construction of “hardware cells”: “previously designed circuit components or structure that have specific physical and functional characteristics used as building blocks for implementing an ASIC to be manufactured.” See Ricoh Opening Claim Construction Br., at pp. 23-24. The technology sensitive modules produced by the DAA meet this construction.

Even assuming, *arguendo*, the technology sensitive modules produced by the DAA are determined not to be hardware cells, then elements [3] and [6] are still met by the combination of the DAA and the module binder, which together produce the claimed hardware cells in two steps: the DAA first produces technology sensitive modules, then the module binder binds each of these modules to a hardware cell, see Kowalski85, at p. 37:

Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint.

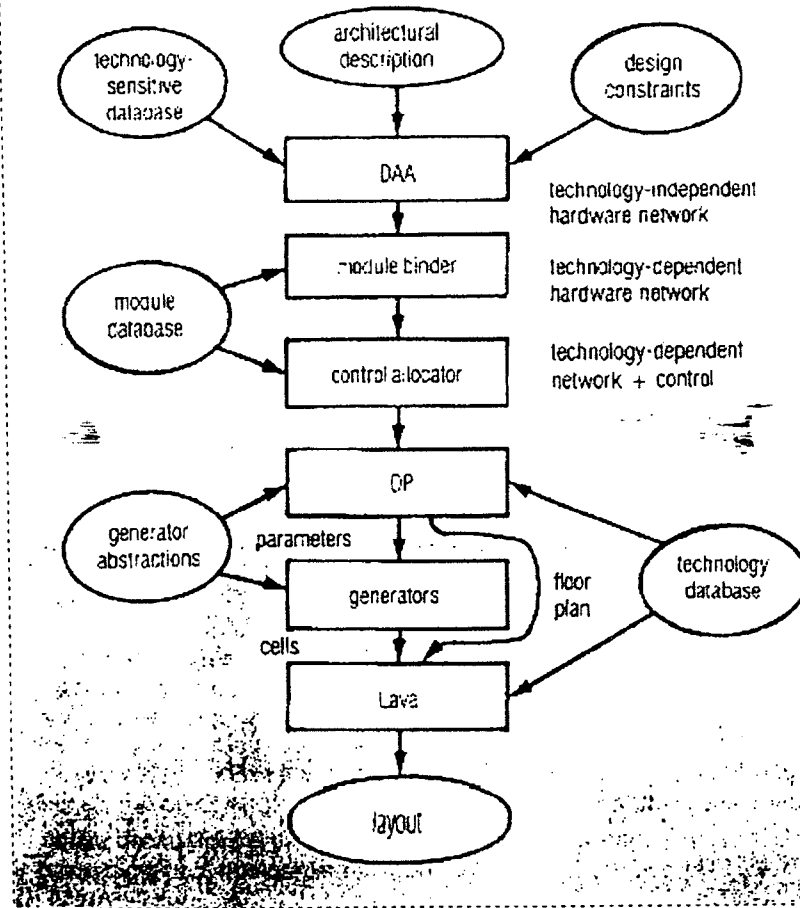
Significantly, there is nothing in the broadest reasonable interpretation of the claim language that requires the hardware cells to be produced in a single step. And Ricoh has admitted that the broadest reasonable interpretation of elements [3] and [6] includes conventional algorithmic software. Therefore, there is nothing that prevents the DAA, an expert system in the traditional sense that includes rules embodying the expert knowledge of VLSI designers, from being combined with the module binder, algorithmic software that binds the technology dependent modules of the DAA into hardware cells, for purposes of satisfying elements [3] and [6].

The following claim charts further demonstrate that Kowalski 85, including the inherent teachings of Kowalski84, anticipates each of claims 13-17 under 35 U.S.C. §102(b):

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising:	<ul style="list-style-type: none"> Figure 1 of Kowalski85, reproduced below, illustrates a multi-stage process that uses a computer to assist in the design of a special purpose VLSI chip, <i>see</i> Kowalski85, p. 34:

'432 PATENT

TEACHINGS OF KOWALSKI85 AND KOWALSKI84



- The process employs at least one computer aided design (CAD) tool for automatically performing one or more stages of the process, *see Kowalski85, p. 33*:

We believe the only solution to this problem is to build new CAD tools that are capable of automatically performing more of the synthesis process.

[element 1]
storing a set of
definitions of
architecture
independent
actions and

- A compiler, a program stored and executing on a computer, stores predefined operators that are used in preparing a dataflow graph representation from an ISPS source file. *See Kowalski85, p. 36.*
- The predefined operators, also called VT operators, form the nodes of the graph; they are translations of the architectural independent actions and conditions in the ISPS source file. *See Kowalski84, p. 49:*

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
conditions;	<div data-bbox="431 317 1414 1052" style="border: 1px dashed black; padding: 10px;"> <p>The ISPS description is compiled into a VT^{17, 18} data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called <i>operators</i> and correspond to operations that take certain values as input and produce new values as output. They are translations of the ISPS unary and binary operations, operations that change or access fields in words or words in arrays, control operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called <i>outnodes</i> and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called <i>VT-bodies</i>, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops.</p> </div> <ul style="list-style-type: none"> • The ISPS source file specifies architecture independent actions and conditions, <i>see</i> Kowalski85, at p. 34 <div data-bbox="703 1220 1163 1591" style="border: 1px dashed black; padding: 10px; margin-top: 20px;"> <p>We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS.</p> </div>
[element 2] storing data	<ul style="list-style-type: none"> • There are two such databases taught by Kowalski85 that collectively or individually meet this limitation. First, there is the database used by the DAA that stores data

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;	<p>describing a set of technology sensitive modules, <i>see</i> Kowalski85, p. 35, which are hardware cells under the broadest reasonable interpretation:</p> <div data-bbox="702 384 1153 863" style="border: 1px dashed black; padding: 5px;"> <p>ISPS,⁹ the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs.</p> </div> <ul style="list-style-type: none"> • Second, there is the database used by the module binder that stores data describing a set of available integrated hardware cells, <i>see</i> Kowalski85, p. 37: <div data-bbox="702 999 1153 1465" style="border: 1px dashed black; padding: 5px;"> <p>Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint.</p> </div>
[element 3] storing in an expert system knowledge base a set of rules for selecting hardware cells	<ul style="list-style-type: none"> • A "cornerstone" of the approach described in Kowalski85 is the use of knowledge-based expert systems, <i>see</i> Kowalski85, p. 34: <div data-bbox="678 1602 1197 1732" style="border: 1px dashed black; padding: 5px;"> <p>A cornerstone of our hardware-synthesis approach is the use of knowledge-based expert systems. Such</p> </div> <ul style="list-style-type: none"> • For example, the DAA, a knowledge-based expert system, stores a set of rules for

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
to perform the actions and conditions;	<p data-bbox="426 289 1468 394">selecting technology sensitive modules that perform each of the specified actions or conditions, wherein each rule embodies the knowledge of expert VLSI chip designers and is stored in an expert system knowledge base, <i>see</i> Kowalski85, p. 36:</p> <div data-bbox="685 426 1164 667" style="border: 1px dashed black; padding: 5px;"> <p data-bbox="697 443 1164 667">DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize an architectural implementation from an algorithmic description with constraints. The de-</p> </div> <ul style="list-style-type: none"> <li data-bbox="388 703 1468 808">• Through application of these rules, the DAA maps the architectural independent actions or conditions into technology sensitive modules, linking paths and control logic, and then it optimizes the resulting design, <i>see</i> Kowalski85, p. 34: <div data-bbox="726 840 1126 1255" style="border: 1px dashed black; padding: 5px;"> <p data-bbox="735 856 1126 1255">ISPS,⁹ the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs.</p> </div> <ul style="list-style-type: none"> <li data-bbox="388 1291 1468 1365">• The rules in the DAA are described in Chapter 5 of Kowalski84, and the type of rules summarized in Table 14 of Kowalski84, at p. 63, as follows:

'432 PATENT

TEACHINGS OF KOWALSKI85 AND KOWALSKI84

Table 14. RULES BY FUNCTION AND KNOWLEDGE TYPE

Section	Rules	Firings	Design	Context	Setup	Cleanup	Input	List	Test
Total DAA	314	8543	151	18	70	33	9	14	19
Service Functions	88	4901	70	7	0	3	0	4	4
Control management	12	1981	0	7	0	1	0	4	0
Module management	21	722	19	0	0	2	0	0	0
Port management	34	732	32	0	0	0	0	0	2
Link management	21	1466	19	0	0	0	0	0	2
Global Allocation	78	104	7	0	70	1	0	0	0
Memories	4	13	3	0	0	1	0	0	0
Registers	2	10	2	0	0	0	0	0	0
Constants	1	10	1	0	0	0	0	0	0
Controller	1	1	1	0	0	0	0	0	0
Technology database	70	70	0	0	70	0	0	0	0
Value Trace Allocation	55	1468	25	0	0	13	9	6	2
Initialization	(See Cleanup)								
Operator assignment	9	322	4	0	0	3	0	2	0
Operator trimming	13	64	0	0	0	2	9	0	2
Temporary registers	2	107	1	0	0	1	0	0	0
Register functions	6	18	6	0	0	0	0	0	0
ALU functions	25	957	14	0	0	7	0	4	0
SCS Allocation	47	1610	21	6	0	9	0	0	11
Register stability	2	173	4	0	0	1	0	0	3
Register allocating	9	26	4	2	0	0	0	0	2
Module stability	5	6	5	0	0	0	0	0	0
Module allocating	18	93	8	4	0	0	0	0	6
Cleanup	8	1312	0	0	0	8	0	0	0
Global Improvements	46	460	28	5	0	7	0	4	2
Unreferenced	6	58	5	0	0	1	0	0	0
Multiplexer cleanup	7	14	5	0	0	1	0	1	0
Multiple fan-out cleanup	4	12	0	3	0	1	0	0	0
Bus utilization	29	376	18	2	0	4	0	3	2
Final cleanup	(See Unreferenced and Multiplexer cleanup)								

- Each of the rules is written in an antecedent IF-THEN-ELSE form, *see* Kowalski84, p. 28:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
	<div data-bbox="396 315 1453 1249" style="border: 1px dashed black; padding: 10px;"> <p data-bbox="454 346 1329 514">3.3.2 The rule memory. The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:</p> <p data-bbox="528 546 1247 703">IF: the most current active context is to create a link and the link should go from a source port to a destination port and the module of the source port is not a multiplexer and there is a link from another module to the same destination port and this other module is not a multiplexer</p> <p data-bbox="536 703 1280 840">THEN: create a multiplexer module and connect the multiplexer to the destination port and connect the source port and destination port link to the multiplexer and move the other link from the destination port to the multiplexer</p> <p data-bbox="479 850 1346 966">This rule recognizes situations in which a multiplexer needs to be created to connect one port to another.</p> <p data-bbox="487 976 1362 1239">Each subtask in the DAA is associated with a set of rules for carrying out the subtask. An example of a rule for the fourth subtask appeared above. Most of the rules, like the example above, define situations in which a partial design should be extended in some particular way. These rules enable the DAA to synthesize an acceptable design by determining, at each step, whether a certain design extension respects constraints.</p> </div> <ul data-bbox="396 1281 1412 1459" style="list-style-type: none"> • Either the DAA or the combination of the DAA and the module binder meets this element under the broadest reasonable interpretation. As discussed, the module binder is algorithmic software embodying expert knowledge that binds the technology sensitive modules produced by the DAA to technology dependent hardware cells, <i>see</i> Kowalski85, at p. 35:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
	<p data-bbox="700 296 1133 737">• The module binder¹⁰: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either selected from the module database or fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University.</p>
<p data-bbox="178 789 356 1157">[element 4] describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;</p>	<ul style="list-style-type: none"> <li data-bbox="393 789 1438 919">• This limitation is met by Kowalski85 when the ISPS source file is entered by a user. The ISPS source file is an algorithmic description of the functionality of the desired chip written by the designer that is not tied to any particular implementation style, fabrication technology, or structure. <i>see</i> Kowalski85, p. 34: <p data-bbox="720 957 1141 1318" style="border: 1px dashed black; padding: 5px; margin: 10px 0;">We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS.</p> <li data-bbox="393 1398 1430 1535">• In ISPS, a user specifies actions (e.g., data operators that do arithmetic, logical, relational and shift operations on the data) and conditions (e.g., sequential, parallel and conditional constructs that show how the data operators are combined). <i>See</i> Kowalski84, pp. 46-47:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
	<p data-bbox="475 331 748 363">4.1 Algorithmic Representation</p> <p data-bbox="484 384 1364 856">In the CMU/DA system, the input description is written in ISPS,¹⁶ a hardware-description language that is similar in many ways to programming languages such as ALGOL or Pascal. An ISPS description consists of a series of declarations of <i>entities</i>. Some of these are simple <i>carriers</i>, which hold the data being manipulated, similar to variables in a programming language. Other entities are procedures or functions much like the procedures or functions of a programming language. These define how the data is manipulated. The procedures are specified by data operators that do arithmetic, logical, relational, and shift operations on the data, and by sequential, parallel, and conditional constructs that show how the data operators are combined.</p>
<p data-bbox="178 919 365 1402">[element 5] specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and</p>	<ul data-bbox="393 919 1455 1024" style="list-style-type: none"> <li data-bbox="393 919 1455 1024">• The ISPS compiler translates each of the actions and conditions specified in the ISPS source file into a VT operator, a stored definition that forms a node of a directed acyclic graph. See Kowalski84, p. 49: <p data-bbox="450 1035 1409 1749">The ISPS description is compiled into a VT^{17, 18} data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called <i>operators</i> and correspond to operations that take certain values as input and produce new values as output. They are translations of the ISPS unary and binary operations, operations that change or access fields in words or words in arrays, control operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called <i>outnodes</i> and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called <i>VT-bodies</i>, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops.</p>

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
<p>[element 6] selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit,</p>	<ul style="list-style-type: none"> • The DAA, a knowledge-based expert system, first selects, for each of the architecture independent actions and conditions of the algorithmic description, one or more technology sensitive modules, <i>see</i> Kowalski85, p. 35. • The module binder then binds each of these modules to an integrated circuit hardware cell, <i>see</i> Kowalski85, p. 37. • Under the broadest reasonable interpretation, either the DAA or the combination of the DAA and module binder meets this element.
<p>[element 6a] said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and</p>	<ul style="list-style-type: none"> • The DAA, a knowledge-based expert system, applies a set of rules to each of the specified definitions of actions or conditions, wherein each rule embodies the knowledge of expert VLSI chip designers and is stored in an expert system knowledge base, <i>see</i> Kowalski85, p. 36: <div data-bbox="690 1052 1169 1289" style="border: 1px dashed black; padding: 10px; margin: 10px 0;"> <p>DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize an architectural implementation from an algorithmic description with constraints. The de-</p> </div> <ul style="list-style-type: none"> • Through application of these rules, the DAA maps the architectural independent actions or conditions into technology sensitive modules, linking paths and control logic, and then it optimizes the resulting design, <i>see</i> Kowalski85, p. 36. • The rules in the DAA are described in Chapter 5 of Kowalski84, and the types of rules summarized in Table 14 of Kowalski84, at p. 63, as follows:

'432 PATENT

TEACHINGS OF KOWALSKI85 AND KOWALSKI84

Table 14. RULES BY FUNCTION AND KNOWLEDGE TYPE

Section	Rules	Firings	Design	Context	Setup	Cleanup	Input	List	Test
Total DAA	314	8543	151	18	70	33	9	14	19
Service Functions	88	4901	70	7	0	3	0	4	4
Control management	12	1981	0	7	0	1	0	4	0
Module management	21	722	19	0	0	2	0	0	0
Port management	34	732	32	0	0	0	0	0	2
Link management	21	1466	19	0	0	0	0	0	2
Global Allocation	78	104	7	0	70	1	0	0	0
Memories	4	13	3	0	0	1	0	0	0
Registers	2	10	2	0	0	0	0	0	0
Constants	1	10	1	0	0	0	0	0	0
Controller	1	1	1	0	0	0	0	0	0
Technology database	70	70	0	0	70	0	0	0	0
Value Trace Allocation	55	1468	25	0	0	13	9	6	2
Initialization	(See Cleanup)								
Operator assignment	9	322	4	0	0	3	0	2	0
Operator trimming	13	64	0	0	0	2	9	0	2
Temporary registers	2	107	1	0	0	1	0	0	0
Register functions	6	18	6	0	0	0	0	0	0
ALU functions	25	957	14	0	0	7	0	4	0
SCS Allocation	47	1610	21	6	0	9	0	0	11
Register stability	2	173	4	0	0	1	0	0	3
Register allocating	9	26	4	2	0	0	0	0	2
Module stability	5	6	5	0	0	0	0	0	0
Module allocating	18	93	8	4	0	0	0	0	6
Cleanup	8	1312	0	0	0	8	0	0	0
Global Improvements	46	460	28	5	0	7	0	4	2
Unreferenced	6	58	5	0	0	1	0	0	0
Multiplexer cleanup	7	14	5	0	0	1	0	1	0
Multiple fan-out cleanup	4	12	0	3	0	1	0	0	0
Bus utilization	29	376	18	2	0	4	0	3	2
Final cleanup	(See Unreferenced and Multiplexer cleanup)								

- Each of the rules is written in an antecedent IF-THEN-ELSE form, *see* Kowalski84, p. 28:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
	<p data-bbox="520 367 1395 514"><i>3.3.2 The rule memory.</i> The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:</p> <p data-bbox="586 556 1305 703"><i>IF:</i> the most current active context is to create a link and the link should go from a source port to a destination port and the module of the source port is not a multiplexer and there is a link from another module to the same destination port and this other module is not a multiplexer</p> <p data-bbox="594 714 1338 840"><i>THEN:</i> create a multiplexer module and connect the multiplexer to the destination port and connect the source port and destination port link to the multiplexer and move the other link from the destination port to the multiplexer</p> <p data-bbox="536 871 1412 955">This rule recognizes situations in which a multiplexer needs to be created to connect one port to another.</p> <p data-bbox="536 987 1420 1239">Each subtask in the DAA is associated with a set of rules for carrying out the subtask. An example of a rule for the fourth subtask appeared above. Most of the rules, like the example above, define situations in which a partial design should be extended in some particular way. These rules enable the DAA to synthesize an acceptable design by determining, at each step, whether a certain design extension respects constraints.</p> <ul data-bbox="446 1291 1437 1365" style="list-style-type: none"> • The module binder then binds each of the modules of the optimized design to a technology-dependent hardware cell, <i>see</i> Kowalski85, at pp. 35 and 37:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
	<p data-bbox="695 296 1141 737">• The module binder¹⁰: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either selected from the module database or fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University.</p> <p data-bbox="695 779 1141 1220">Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint.</p> <ul data-bbox="389 1262 1438 1325" style="list-style-type: none"> • Under the broadest reasonable interpretation, either the DAA or the combination of the DAA and module binder meets this element.
[element 6b] generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the	<ul data-bbox="389 1371 1447 1577" style="list-style-type: none"> • The result of the module binder is a technology-dependent hardware network, <i>e.g.</i>, a netlist. <i>See</i> Kowalski85, Figure 1, p. 34. • In the '432 Patent, Patent Owner admitted that a technology-dependent network such as this is a netlist. <i>See</i> Col 5:36-37 ("The netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block.").

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
desired function of the integrated circuit and the interconnection requirements therefore.	
14. A process as defined in claim 13, including generating from the netlist the mask data required to produce an integrated circuit having the desired function.	<ul style="list-style-type: none"> In the '432 patent, Patent Owner admitted that commercial systems were available to convert a netlist, such as the technology-dependent hardware network output by the DAA and module binder of Kowalski85, to mask data. <i>See</i> Col 5:40-44 ("Computer-aided designs systems for cell placement and routing are commercially available which will receive netlist data as input and will . . . produce mask data. . . .").
15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.	<ul style="list-style-type: none"> The DAA generates the data paths, also called "ports and links," <i>see</i> Kowalski84, p. 64: <div data-bbox="437 1087 1424 1476" style="border: 1px dashed black; padding: 10px; margin: 10px 0;"> <p>The functional subtasks in the DAA differ in numbers of rules, frequency of use, and type of knowledge. Table 14 shows that the service functions and global allocation subtasks contain over half of the total rules and half the rules that extend the partial design. Furthermore, each of the remaining three subtasks contains about fifteen percent of the total rules and fifteen percent of the rules that extend the partial design. The most frequently used design rules are the port and link management rules, which follows logically because these are the rules that create the data path. These rules are discussed in the following section.</p> </div> The task of allocating data paths is one of the main tasks performed by the DAA, <i>see</i> Kowalski84, p 95:

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84																																																																																																																																																																																																																																																																																																																																
	<div>5.7 Knowledge Summary</div> <div>Chapter 5 has shown how designers partition the design task into four major tasks. These tasks involve global allocation of architectural modules and registers; local allocation of control steps, operators, registers, data paths and control; global allocation of operators and registers; and global improvements to the design. Within these tasks, a dominant goal</div>																																																																																																																																																																																																																																																																																																																																
16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.	<div>As indicated in the above discussion for claim 15, the DAA uses a number of port management and link management rules, summarized in Table 14 of Kowalski84, at p. 63, reproduced below, to "create the data path," see Kowalski84, p. 64:</div> <div>Table 14. RULES BY FUNCTION AND KNOWLEDGE TYPE</div> <table><tr><th>Section</th><th>Rules</th><th>Firings</th><th>Design</th><th>Context</th><th>Setup</th><th>Cleanup</th><th>Input</th><th>List</th><th>Test</th></tr><tr><td>Total DAA</td><td>314</td><td>8543</td><td>151</td><td>18</td><td>70</td><td>33</td><td>9</td><td>14</td><td>19</td></tr><tr><td>Service Functions</td><td>88</td><td>4901</td><td>70</td><td>7</td><td>0</td><td>3</td><td>0</td><td>4</td><td>4</td></tr><tr><td>Control management</td><td>12</td><td>1981</td><td>0</td><td>7</td><td>0</td><td>1</td><td>0</td><td>4</td><td>0</td></tr><tr><td>Module management</td><td>21</td><td>722</td><td>19</td><td>0</td><td>0</td><td>2</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Port management</td><td>34</td><td>732</td><td>32</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td></tr><tr><td>Link management</td><td>21</td><td>1466</td><td>19</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td></tr><tr><td>Global Allocation</td><td>78</td><td>104</td><td>7</td><td>0</td><td>70</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Memories</td><td>4</td><td>13</td><td>3</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Registers</td><td>2</td><td>10</td><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Constants</td><td>1</td><td>10</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Controller</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Technology database</td><td>70</td><td>70</td><td>0</td><td>0</td><td>70</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Value Trace Allocation</td><td>55</td><td>1468</td><td>25</td><td>0</td><td>0</td><td>13</td><td>9</td><td>6</td><td>2</td></tr><tr><td>Initialization</td><td colspan="9">(See Cleanup)</td></tr><tr><td>Operator assignment</td><td>9</td><td>322</td><td>4</td><td>0</td><td>0</td><td>3</td><td>0</td><td>2</td><td>0</td></tr><tr><td>Operator trimming</td><td>13</td><td>64</td><td>0</td><td>0</td><td>0</td><td>2</td><td>9</td><td>0</td><td>2</td></tr><tr><td>Temporary registers</td><td>2</td><td>107</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Register functions</td><td>6</td><td>18</td><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>ALU functions</td><td>25</td><td>957</td><td>14</td><td>0</td><td>0</td><td>7</td><td>0</td><td>4</td><td>0</td></tr><tr><td>SCS Allocation</td><td>47</td><td>1610</td><td>21</td><td>6</td><td>0</td><td>9</td><td>0</td><td>0</td><td>11</td></tr><tr><td>Register stability</td><td>2</td><td>173</td><td>4</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>3</td></tr><tr><td>Register allocating</td><td>9</td><td>26</td><td>4</td><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td></tr><tr><td>Module stability</td><td>5</td><td>6</td><td>5</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Module allocating</td><td>18</td><td>93</td><td>8</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>6</td></tr><tr><td>Cleanup</td><td>8</td><td>1312</td><td>0</td><td>0</td><td>0</td><td>8</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Global Improvements</td><td>46</td><td>460</td><td>28</td><td>5</td><td>0</td><td>7</td><td>0</td><td>4</td><td>2</td></tr><tr><td>Unreferenced</td><td>6</td><td>58</td><td>5</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Multiplexer cleanup</td><td>7</td><td>14</td><td>5</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>Multiple fan-out cleanup</td><td>4</td><td>12</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Bus utilization</td><td>29</td><td>376</td><td>18</td><td>2</td><td>0</td><td>4</td><td>0</td><td>3</td><td>2</td></tr><tr><td>Final cleanup</td><td colspan="9">(See Unreferenced and Multiplexer cleanup)</td></tr></table>	Section	Rules	Firings	Design	Context	Setup	Cleanup	Input	List	Test	Total DAA	314	8543	151	18	70	33	9	14	19	Service Functions	88	4901	70	7	0	3	0	4	4	Control management	12	1981	0	7	0	1	0	4	0	Module management	21	722	19	0	0	2	0	0	0	Port management	34	732	32	0	0	0	0	0	2	Link management	21	1466	19	0	0	0	0	0	2	Global Allocation	78	104	7	0	70	1	0	0	0	Memories	4	13	3	0	0	1	0	0	0	Registers	2	10	2	0	0	0	0	0	0	Constants	1	10	1	0	0	0	0	0	0	Controller	1	1	1	0	0	0	0	0	0	Technology database	70	70	0	0	70	0	0	0	0	Value Trace Allocation	55	1468	25	0	0	13	9	6	2	Initialization	(See Cleanup)									Operator assignment	9	322	4	0	0	3	0	2	0	Operator trimming	13	64	0	0	0	2	9	0	2	Temporary registers	2	107	1	0	0	1	0	0	0	Register functions	6	18	6	0	0	0	0	0	0	ALU functions	25	957	14	0	0	7	0	4	0	SCS Allocation	47	1610	21	6	0	9	0	0	11	Register stability	2	173	4	0	0	1	0	0	3	Register allocating	9	26	4	2	0	0	0	0	2	Module stability	5	6	5	0	0	0	0	0	0	Module allocating	18	93	8	4	0	0	0	0	6	Cleanup	8	1312	0	0	0	8	0	0	0	Global Improvements	46	460	28	5	0	7	0	4	2	Unreferenced	6	58	5	0	0	1	0	0	0	Multiplexer cleanup	7	14	5	0	0	1	0	1	0	Multiple fan-out cleanup	4	12	0	3	0	1	0	0	0	Bus utilization	29	376	18	2	0	4	0	3	2	Final cleanup	(See Unreferenced and Multiplexer cleanup)								
Section	Rules	Firings	Design	Context	Setup	Cleanup	Input	List	Test																																																																																																																																																																																																																																																																																																																								
Total DAA	314	8543	151	18	70	33	9	14	19																																																																																																																																																																																																																																																																																																																								
Service Functions	88	4901	70	7	0	3	0	4	4																																																																																																																																																																																																																																																																																																																								
Control management	12	1981	0	7	0	1	0	4	0																																																																																																																																																																																																																																																																																																																								
Module management	21	722	19	0	0	2	0	0	0																																																																																																																																																																																																																																																																																																																								
Port management	34	732	32	0	0	0	0	0	2																																																																																																																																																																																																																																																																																																																								
Link management	21	1466	19	0	0	0	0	0	2																																																																																																																																																																																																																																																																																																																								
Global Allocation	78	104	7	0	70	1	0	0	0																																																																																																																																																																																																																																																																																																																								
Memories	4	13	3	0	0	1	0	0	0																																																																																																																																																																																																																																																																																																																								
Registers	2	10	2	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																								
Constants	1	10	1	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																								
Controller	1	1	1	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																								
Technology database	70	70	0	0	70	0	0	0	0																																																																																																																																																																																																																																																																																																																								
Value Trace Allocation	55	1468	25	0	0	13	9	6	2																																																																																																																																																																																																																																																																																																																								
Initialization	(See Cleanup)																																																																																																																																																																																																																																																																																																																																
Operator assignment	9	322	4	0	0	3	0	2	0																																																																																																																																																																																																																																																																																																																								
Operator trimming	13	64	0	0	0	2	9	0	2																																																																																																																																																																																																																																																																																																																								
Temporary registers	2	107	1	0	0	1	0	0	0																																																																																																																																																																																																																																																																																																																								
Register functions	6	18	6	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																								
ALU functions	25	957	14	0	0	7	0	4	0																																																																																																																																																																																																																																																																																																																								
SCS Allocation	47	1610	21	6	0	9	0	0	11																																																																																																																																																																																																																																																																																																																								
Register stability	2	173	4	0	0	1	0	0	3																																																																																																																																																																																																																																																																																																																								
Register allocating	9	26	4	2	0	0	0	0	2																																																																																																																																																																																																																																																																																																																								
Module stability	5	6	5	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																								
Module allocating	18	93	8	4	0	0	0	0	6																																																																																																																																																																																																																																																																																																																								
Cleanup	8	1312	0	0	0	8	0	0	0																																																																																																																																																																																																																																																																																																																								
Global Improvements	46	460	28	5	0	7	0	4	2																																																																																																																																																																																																																																																																																																																								
Unreferenced	6	58	5	0	0	1	0	0	0																																																																																																																																																																																																																																																																																																																								
Multiplexer cleanup	7	14	5	0	0	1	0	1	0																																																																																																																																																																																																																																																																																																																								
Multiple fan-out cleanup	4	12	0	3	0	1	0	0	0																																																																																																																																																																																																																																																																																																																								
Bus utilization	29	376	18	2	0	4	0	3	2																																																																																																																																																																																																																																																																																																																								
Final cleanup	(See Unreferenced and Multiplexer cleanup)																																																																																																																																																																																																																																																																																																																																
17. A process	As indicated in the above discussion for claim 15, the DAA uses a number of port management and link management rules, summarized in Table 14 of Kowalski84, at p. 63, reproduced below, to "create the data path," see Kowalski84, p. 64:																																																																																																																																																																																																																																																																																																																																

'432 PATENT	TEACHINGS OF KOWALSKI85 AND KOWALSKI84
as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.	<p>paths, <i>see</i> Kowalski85, p. 37:</p> <div data-bbox="702 357 1153 676" style="border: 1px dashed black; padding: 10px;"> <p>Control allocator. The control allocator translates the technology-independent control specification, generated by the DAA, into a controller implementation in a chosen technology. A style-independent part determines the control signals that drive the data path and a style-dependent part designs the sequencer. This</p> </div>

VI. STATEMENT POINTING OUT SUBSTANTIAL NEW QUESTION OF PATENTABILITY

Neither of the Kowalski85 and Kowalski84 prior art references were of record during the original prosecution of the '432 patent. Further, for the reasons discussed, these references are more pertinent to the subject matter claimed in claims 13-17 of the '432 patent than the Darringer and Nash references that were applied during the original prosecution. As demonstrated above, a substantial new question of patentability is raised because each of these claims is anticipated by Kowalski85 (in which Kowalski84 is inherent) under 35 U.S.C. §102(b), particularly in light of Ricoh's position in the district court litigation as to the absence in the claims of the precise features—"flowchart editor" and "expert system"—that were determined by the Examiner to be key to patentability. In the alternative, a substantial new question of patentability is raised because each of these claims is obvious over Kowalski85 in view of Kowalski84 under 35 U.S.C. §103 because Kowalski85 cites to Kowalski84, providing the suggestion to combine.

VII. AN ADDITIONAL SUBSTANTIAL NEW QUESTION OF PATENTABILITY IS RAISED BY CONSIDERING DARRINGER IN VIEW OF KOWALSKI85

An additional substantial new question of patentability under 35 U.S.C. § 103 is raised by consideration of the Darringer reference in light of Kowalski85. In particular, claims 13-17 are unpatentable over the combination of Darringer with Kowalski85, as the obvious modification of

Darringer as taught and suggested by Kowalski85 discloses all of the features set forth in claims 13-17.

The Patent Owner admitted during prosecution that Darringer “does disclose a method and system for automatic logic design and it is known in the art of automatic layout to utilize cell libraries of circuit components.” Paper No. 6 at 9. The Patent Owner distinguished Darringer solely on the basis that Darringer did not disclose an “architecture independent” functional specification input or a rule based expert system for automatic logic synthesis. *Id.*

However, Kowalski85 clearly taught those of ordinary skill in the art at the time of the alleged invention a CAD tool that automatically performs more of the synthesis process by allowing the designer to input a high level, architecture independent functional circuit description as contrasted with the register transfer level input taught in Darringer to allow “the evaluation of different technologies in order to choose the one best suited to the particular problem.” Kowalski85 at 34. Kowalski85 further taught those of ordinary skill in the art of hardware synthesis “the use of knowledge-based expert systems.” *Id.*

At the time of the alleged invention, it would have been obvious to one of ordinary skill in the art to have added the architecture independent descriptive input to the Darringer system as taught and suggested by Kowalski85, to thereby allow the system “to choose the one best suited to the particular problem” as taught by Kowalski85 (*see* p. 34). Further, it would have been obvious to one of ordinary skill in the art at the time of the alleged invention to have incorporated into Darringer the use of a knowledge-based expert system as taught by Kowalski85, so as to allow the system to “make decisions based on knowledge, expressed as rules, obtained from expert designers” as also taught by Kowalski85. Kowalski85 at 34.


Inasmuch as the Patent Owner acknowledged that Darringer discloses a CAD system for integrated circuits that is basically the same as that disclosed in the ‘432 patent, and distinguished the claims from the Darringer reference solely on the basis of two features (both taught by Kowalski85), the obvious modification of the Darringer reference as clearly taught and

suggested by Kowalski⁸⁵ indisputably results in a CAD design process as set forth in claims 13-17.

VIII. CONCLUSION

For all the foregoing reasons, this Request should be granted and reexamination ordered pursuant to 37 C.F.R. §1.525. Please charge any fee insufficiency or credit any fee overpayment to Novak Druce Deposit Account No. 14-1437.

Respectfully submitted,


Donald J. Quigg
Reg. No. 16,030

Of Counsel:

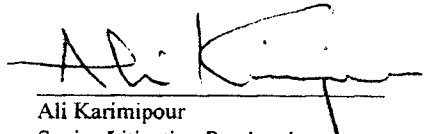
Vincent M. DeLuca
Registration No. 32,408

Novak Druce DeLuca & Quigg LLP
1300 Eye St., N.W.
Suite 400 East Tower
Washington, DC 20005
(202) 659-0100 (telephone)
(202) 659-0105 (facsimile)

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a copy of this REQUEST FOR REEXAMINATION with all exhibits and attachments and supporting documentation, has been served by **FIRST CLASS MAIL**, postage prepaid, this the 17th day of January 2006 upon the patent owner and upon all parties to the current litigations involving the patent at issue, as follows:

<u>Ricoh Company, Ltd.</u> Altshuler Berzon Nussbaum Rubin & Demain 177 Post Street, Suite 300 San Francisco, CA 94108	<u>Aeroflex Incorporated</u> <u>AMI Semiconductor, Inc.</u> <u>Matrox Electronic Systems, Ltd.</u> HOWREY SIMON ARNOLD & WHITE, LLP 301 Ravenswood Avenue Menlo Park, California 94025
<u>Synopsys, Inc.</u> HOWREY SIMON ARNOLD & WHITE, LLP 301 Ravenswood Avenue Menlo Park, California 94025	


Ali Karimipour
Senior Litigation Paralegal

EXHIBIT

39



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/007,879	01/17/2006	4922432	8089.002	1132

24998 7590 02/24/2006

DICKSTEIN SHAPIRO MORIN & OSHINSKY LLP
 2101 L Street, NW
 Washington, DC 20037

EXAMINER

ART UNIT

PAPER NUMBER

DATE MAILED: 02/24/2006

Please find below and/or attached an Office communication concerning this application or proceeding.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

Novak, Druce, DeLuca & Quigg, LLP
1300 Eye St. NW Suite 400 East Tower
Washington DC 20005

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. 90/007,879.

PATENT NO. 4922432.

ART UNIT 3992.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

**Order Granting / Denying Request For
Ex Parte Reexamination**

Control No.

90/007,879

Patent Under Reexamination

4922432

Examiner

St. John Courtenay III

Art Unit

3992

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

The request for *ex parte* reexamination filed 17 January 2006 has been considered and a determination has been made. An identification of the claims, the references relied upon, and the rationale supporting the determination are attached.

Attachments: a) ☐ PTO-892, b) ☒ PTO-1449, c) ☐ Other: _____

1. ☒ The request for *ex parte* reexamination is GRANTED.

RESPONSE TIMES ARE SET AS FOLLOWS:

For Patent Owner's Statement (Optional): TWO MONTHS from the mailing date of this communication (37 CFR 1.530 (b)). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).**

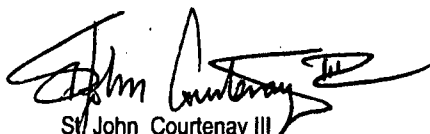
For Requester's Reply (optional): TWO MONTHS from the date of service of any timely filed Patent Owner's Statement (37 CFR 1.535). **NO EXTENSION OF THIS TIME PERIOD IS PERMITTED.** If Patent Owner does not file a timely statement under 37 CFR 1.530(b), then no reply by requester is permitted.

2. ☐ The request for *ex parte* reexamination is DENIED.

This decision is not appealable (35 U.S.C. 303(c)). Requester may seek review by petition to the Commissioner under 37 CFR 1.181 within ONE MONTH from the mailing date of this communication (37 CFR 1.515(c)). **EXTENSION OF TIME TO FILE SUCH A PETITION UNDER 37 CFR 1.181 ARE AVAILABLE ONLY BY PETITION TO SUSPEND OR WAIVE THE REGULATIONS UNDER 37 CFR 1.183.**

In due course, a refund under 37 CFR 1.26 (c) will be made to requester:

- a) ☐ by Treasury check or,
b) ☐ by credit to Deposit Account No. _____, or
c) ☐ by credit to a credit card account, unless otherwise notified (35 U.S.C. 303(c)).


St. John Courtenay III
Primary Examiner
Art Unit: 3992

cc:Requester (if third party requester)

Application/Control Number: 90/007,879
Art Unit: 3992

Page 2

Response to Request for *ex parte* Reexamination

1. Reexamination has been requested for claims 13-17 of U.S. Patent number 4,922,432 ('432 patent).
2. A substantial new question of patentability affecting at least claim 13 of United States Patent number 4,922,432 is raised by the request for *ex parte* reexamination.
3. The '432 patent is currently assigned to:

RICOH COMPANY, LTD.
OHTA-KU
3-6, 1-CHOME NAKAMAGOME
TOKYO, JAPAN
4. The '432 patent application filing date was Jan. 13, 1988 and the patent issued on May 1, 1990. There is no claim to priority of record.
5. In the request for reexamination, the requester alleges that '432 patent claims 13-17 are anticipated under 35 U.S.C. §102 in light of the following references:
 - **T.J. KOWALSKI**, D.J. Gelger, W. H. Wolf, W. Fichtner, The VLSI Design Automation Assistant: From Algorithms to Silicon, IEEE Design & Test, pp. 33-43 (1985). (i.e., "**KOWALSKI-85**")
 - **Thaddeus Julius KOWALSKI**, The VLSI Design Automation Assistant: A Knowledge- Based Expert System, Carnegie-Mellon University PhD Thesis, April 1984. (i.e., "**KOWALSKI-84**")

Application/Control Number: 90/007,879
Art Unit: 3992

Page 3

6. The newly cited **KOWALKSKI-85** and **KOWALKSKI-84** references were not of record in the file of the '432 patent and are not cumulative to the art of record in the original file.

7. It is agreed that the **KOWALKSKI-85** reference (including the inherent teachings of **KOWALKSKI-84**) would have been considered important by a reasonable Examiner in deciding whether or not at least claim 13 was patentable, for the reasons discussed *infra*.

8. **As per independent claim 13:**

The **KOWALKSKI-85** reference (including the inherent teachings of **KOWALKSKI-84**) teaches a computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising:

- storing a set of definitions of architecture independent actions and conditions [see e.g., Kowalski85, p. 36: A compiler, a program stored and executing on a computer, stores predefined operators that are used in preparing a dataflow graph representation from an ISPS source file. The predefined operators, also called VT operators, form the nodes of the graph; they are translations of the architectural independent actions and conditions in the ISPS source file. See also Kowalski84, p. 49: The ISPS description is compiled into a VT data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called *operators* and correspond to operations that take certain values as input and produce new

Application/Control Number: 90/007,879
 Art Unit: 3992

Page 4

values as output. They are translations of the ISPS unary and binary operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called *outnodes* and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called VT-bodies, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops"; The ISPS source file specifies architecture independent actions and conditions, see Kowalski85, at p. 34: "We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS"];

- storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set [e.g., There are two databases taught by Kowalski85 that collectively or individually meet this limitation. Firstly, there is the database used by the DAA that stores data describing a set of technology sensitive modules, see Kowalski85, p. 35, which are hardware cells under the broadest reasonable interpretation: "the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs"; Secondly, there is the database used by the module binder that stores data describing a set of available integrated hardware cells, see Kowalski85, page 37: "Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder

Application/Control Number: 90/007,879
 Art Unit: 3992

Page 5

may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint”];

- storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions [e.g., A “cornerstone of the approach described in Kowalski85 is the use of knowledge based expert systems, see Kowalski85, page 34: “A cornerstone of our hardware-synthesis approach is the use of knowledge-based expert systems.” For example, the DAA, a knowledge-based expert system, stores a set of rules for selecting technology sensitive modules that perform each of the specified actions or conditions, wherein each rule embodies the knowledge of expert VLSI chip designers and is stored in an expert system knowledge base, see Kowalski85, page 36: “DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize and architectural implementation from an algorithmic description with constraints.”; Through application of these rules, the DAA maps the architectural independent actions or condition into technology sensitive modules, linking paths and control logic, and then it optimizes the resulting design, see Kowalski85, page 34: “the DAA uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about the trade-offs particular to the target technology; the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. We are currently developing a way to create pipeline-style designs”; The rules in the DAA are describe in Chapter 5, of Kowalski84, at page 63. Each of the rules is written in an antecedent IF-THEN-ELSE form, see Kowalski84, page 28. Either the DAA or the combination of the DAA and the module binder meets this element under the broadest reasonable interpretation. As discussed, the module binder is algorithmic software embodying expert knowledge that binds the technology sensitive modules produced by the DAA to technology dependent hardware cells, see Kowalski85, at page 35: “The module binder: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either

Application/Control Number: 90/007,879
 Art Unit: 3992

Page 6

selected from the module database of fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University."];

- describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions [This limitation is met by Kowalski85 when the ISPS source file is entered by a user. The ISPS source file is an algorithmic description of the functionality of the desired chip written by the designer that is not tied to any particular implementation style, fabrication technology, or structure. See Kowalski85, p. 34: "We view hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer first specifies the chip's functionality as an algorithm. The algorithmic description is not tied to a particular implementation style, such as parallel or serial, sequential or pipelined. Similarly, it is not restricted to a fabrication technology, such as TTL, ECL, NMOS, or CMOS"; In ISPS, a user specifies actions (e.g., data operators that do arithmetic, logical, relational and shift operations on the data) and conditions (e.g., sequential, parallel and conditional constructs that show how the data operators are combined). See Kowalski84, pp. 46-47: "4.1 Algorithmic Representation - In the CMU/DA system, the input description is written in ISPS, a hardware-description language that is similar in many ways to programming languages such as ALGOL or Pascal. An ISPS description consists of a series of declarations of *entities*. Some of these are simple *carriers*, which hold the data being manipulated, similar to variables in a programming language. Other entities are procedures or functions much like the procedures or function of a programming language. These define how the data is manipulated. The procedures are specified by data *operators* that do arithmetic, logical, relational, and shift operations on the data, and by sequential, parallel, and conditional constructs that show how the data operators are combined."];

Application/Control Number: 90/007,879

Page 7

Art Unit: 3992

- specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed [e.g., The ISPS compiler translates each of the actions and conditions specified in the ISPS source file into a VT operator, a stored definition that forms a node of a directed acyclic graph. See Kowalski84, p. 49: "The ISPS description is compiled into a VT data-flow representation, which makes it easier to recognize and implement design decisions by synthesis programs. The VT is a directed acyclic graph, DAG, similar in nature to those used in optimizing compilers, with the addition of control constructs to allow conditionals and subroutines in the VT. The nodes in this graph are called *operators* and correspond to operations that take certain values as input and produce new values as output. They are translations of the ISPS unary and binary operations, operations that change or access fields in worlds or worlds in arrays, control operations such as procedure or block invocation, and conditional branches. The arcs connecting the nodes are called *outnodes* and represent the generation or use of data values. They are translations of the ISPS carriers and the temporary carriers needed to pass results from one operator to another. The graph is partitioned into subgraphs called *VT-bodies*, corresponding to a set of operations that can be evoked, entered, or left as a unit. These subgraphs are translations of ISPS procedures, labeled blocks and loops."]; and,
- selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit [e.g., The DAA, a knowledge-based expert system, first selects, for each of the architecture Independent actions and conditions of the algorithmic description, one or more technology sensitive modules, see Kowalski85, p. 35. The module binder then binds each of these modules to an integrated circuit hardware cell, see Kowalski85, p. 37. Under the broadest reasonable interpretation, either the DAA or the combination of the DAA and module binder meets this element.],

Application/Control Number: 90/007,879
 Art Unit: 3992

Page 8

- said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base [e.g., The DAA, a knowledge-based expert system, applies a set of rules to each of the specified definitions of actions or conditions, wherein each rule embodies the knowledge of expert VLSI chip designers and is stored in an expert system knowledge base, see Kowalski85, p. 36: "DAA knowledge. The DAA is a knowledge-based expert system that uses a database of more than 500 rules to synthesize an architectural implementation from an algorithmic description with constraints." Through application of these rules, the DAA maps the architectural independent actions or conditions into technology sensitive modules, linking paths and control logic, and then it optimizes the resulting design, see Kowalski85, p. 36. The rules in the DAA are described in Chapter 5 of Kowalski84, and the types of rules summarized in Table 14 of Kowalski84, at p. 63. Each of the rules is written in an antecedent IF-THEM-ELSE form, see Kowalski84, page 28. The module binder then binds each of the modules of the optimized design to a technology-dependent hardware cell, see Kowalski85, at pages 35 and 37: (page 35) "The module binder: an algorithmic program that builds the modules specified by the DAA from components in the target technology. The technology-dependent modules are either selected from the module database or fabricated from simpler, equivalent database entries. The selection process is guided by user-supplied constraints on area, power, and speed. The module binder is the master's thesis project of E. Dirkes of Carnegie-Mellon University." (page 37) "Module binding. The module binder chooses physically realizable, technology-dependent cells to implement the modules in the DAA's design. The cells are chosen from the module database according to user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint, or a carry-lookahead adder when performance is the major design constraint." Under the broadest reasonable interpretation, either the DAA or the combination of the DAA and module binder meets this element.], and,

Application/Control Number: 90/007,879
Art Unit: 3992

Page 9

- generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor [e.g., The result of the module binder is a technology-dependent hardware network, e.g., a netlist. See Kowalski85, Figure 1, p. 34. In the '432 Patent, the Patent Owner admitted that a technology-dependent network such as this is a "netlist." See Col. 5, lines 36-37 ("the netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block.")].

Application/Control Number: 90/007,879
Art Unit: 3992

Page 10

Conclusion

9. All claims are subject to reexamination.

10. Extensions of time under 37 C.F.R. §1.136(a) will not be permitted in this proceeding because the provisions of 37 C.F.R. §1.136 apply only to "an Applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. §305 requires that *ex parte* reexamination proceedings "will be conducted with **special dispatch**" (37 C.F.R. §1.550(a)). Extensions of time in *ex parte* reexamination proceedings are provided for in 37 C.F.R. §1.550(c).

11. The Patent Owner is reminded of the continuing responsibility under 37 C.F.R. § 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent number 4,922,432 throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Application/Control Number: 90/007,879
Art Unit: 3992

Page 11

How to Communicate with the USPTO

ALL correspondence relating to this *ex partes* reexamination proceeding should be directed as follows:

Please mail any communications to:

Attn: Mail Stop "Ex Parte Reexam"
Central Reexamination Unit
Commissioner for Patents
P. O. Box 1450
Alexandria VA 22313-1450

Please FAX any communications to:

(571) 273-9900
Central Reexamination Unit

Please hand-deliver any communications to:

Customer Service Window
Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

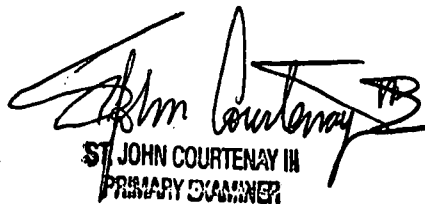
Signed:

St. John Courtenay III (Monday - Friday 9:00 AM - 5:30 PM)
Primary Examiner
Central Reexamination Unit 3992
(571) 272-3761

Conferences:

Michael SPRE-CRU-3992

Major Bonenluch, primary Examiner - CRU-3992


ST. JOHN COURTENAY III
PRIMARY EXAMINER

PTO/SB/088 (07-05)

Approved for use through 07/31/2006. OMB 0651-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Substitute for form 1448/PTO INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Use as many sheets as necessary)		Complete if Known			
		Application Number	Reexam of Pat. No. 4,922,432		
		Filing Date	January 13, 1988		
		First Named Inventor	Hideaki KOBAYASHI		
		Art Unit			
		Examiner Name			
Sheet	1	of	1	Attorney Docket Number	8089.002

NON PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No. ¹	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T ²
SC		T.J. KOWALKSKI, D.J. Geiger, W. H. Wolf, W. Fichtner, The VLSI Design Automation Assistant: From Algorithms to Silicon, IEEE Design & Test, pp. 33-43 (1985)	
SC		Thaddeus Julius KOWALSKI, The VLSI Design Automation Assistant: A Knowledge-Based Expert System, Carnegie-Mellon University PhD Thesis, April 1984	

Examiner Signature	S. COURTENAY	Date Considered	2-16-2006
-----------------------	--------------	--------------------	-----------

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

¹ Applicant's unique citation designation number (optional). ² Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.88. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 (1-800-786-9199) and select option 2.

EXHIBIT

40

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

Attorney Docket: 3868-2

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS,

the specification of which

(check one)

☒ [X] is attached hereto.

☐ [] was filed on _____ as

Application Serial No. _____

and was amended on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

English Language Declaration

Prior Foreign Application(s)			Priority Claimed	
<u>None</u>			[]	[]
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
<u> </u>	<u> </u>	<u> </u>	[]	[]
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
<u> </u>	<u> </u>	<u> </u>	[]	[]
(Number)	Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

<u>None</u>		
(Appln. Serial No.)	(Filing Date)	(Status)
		(patented/pending/aban.)
<u> </u>	<u> </u>	<u> </u>
(Appln. Serial No.)	(Filing Date)	(Status)
		(patented/pending/aban.)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

English Language Declaration

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith (list name and registration number).

Raymond O. Linker, Jr.
Registration No. 26,419

Send correspondence to: Raymond O. Linker, Jr.
Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234

Direct telephone calls to: (name and telephone number)

Raymond O. Linker, Jr.
(704) 377-1561

Full name of sole inventor: Hideaki Kobayashi

Inventor's
Signature: Hideaki Kobayashi Date: January 12, 1988

Residence: Columbia, South Carolina

Citizenship: Japan

Post Office Address: 1401 Main Street
Columbia, South Carolina 29201

Full name of second inventor: Masahiro Shindo

Inventor's
Signature: Masahiro Shindo Date: Dec. 26, 1987

Residence: Osaka, Japan

Citizenship: Japan

Post Office Address: 13-1, Himemuro-cho
Ikeda, Osaka, 563
Japan



UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
07/43,821			

EXAMINER	
TRANS	
ART UNIT	PAPER NUMBER
234	8

DATE MAILED:

EXAMINER INTERVIEW SUMMARY RECORD

All participants (applicant, applicant's representative, PTO personnel)

(1) Mr. F. Licker (2) _____

(3) _____ (4) _____

Date of interview Oct 19, 1989

Type: ☐ Telephonic ☒ Personal (copy is given to ☐ applicant ☒ applicant's representative).

Exhibit shown or demonstration conducted: ☐ Yes ☒ No. If yes, brief description: _____

Agreement ☒ was reached with respect to some or all of the claims in question. ☐ was not reached.

Claims discussed: Claims 1, 5, 15, 18, 20 and 27

Identification of prior art discussed: Darringer et al. (U.S. Patent 4,903,435).

Description of the general nature of what was agreed to if an agreement was reached, or any other comments: It is agreed that the features "flowchart editor" and "expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit" are patentable distinct from the reference identified above. Thus, applicant's attorney will amend the claims to include these features.
(A fuller description, if necessary, and a copy of the amendments, if available, which the examiner agreed would render the claims allowable must be attached. Also, where no copy of the amendments which would render the claims allowable is available, a summary thereof must be attached.)

Unless the paragraphs below have been checked to indicate to the contrary, A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION IS NOT WAIVED AND MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW (e.g., items 1-7 on the reverse side of this form). If a response to the last Office action has already been filed, then applicant is given one month from this interview date to provide a statement of the substance of the interview.

☐ It is not necessary for applicant to provide a separate record of the substance of the interview.

☐ Since the examiner's interview summary above (including any attachments) reflects a complete response to each of the objections, rejections and requirements that may be present in the last Office action, and since the claims are now allowable, this completed form is considered to fulfill the response requirements of the last Office action.

Examiner's Signature

PTOL-413 (REV. 1-84)

ORIGINAL FOR INSERTION IN RIGHT HAND FLAP OF FILE WRAPPER

EXHIBIT

41



PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Hideaki Kobayashi and
Masahiro Shindo
Filed: Concurrently Herewith
For: KNOWLEDGE BASED METHOD AND
APPARATUS FOR DESIGNING
INTEGRATED CIRCUITS USING
FUNCTIONAL SPECIFICATIONS

The Honorable Commissioner of
Patents and Trademarks
Washington, D.C. 20231

January 13, 1988

INFORMATION DISCLOSURE STATEMENT

Sir:

The patents listed on the attached form PTO 1449 were considered by applicants and applicants' attorney in the preparation of this application. While these patents may be of some general interest as background information in the field of artificial intelligence technology, they are not believed to be material to the patentability of the present invention. They are submitted herewith for consideration by the Examiner pursuant to Rule 60.

The excerpt from the textbook An Engineering Approach to Digital Design is mentioned in applicants' specification. A copy is enclosed for consideration by the Examiner and to complete the record.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
(704) 377-1561
ROLjr:klc
Our File 3868-2

RCL000064

EXHIBIT

42



5

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

H. Kobayashi & M. Shindo
Serial No. 143,821
Filed: January 13, 1988
For: *Knowledge Based Method
and Apparatus For Designing
Integrated Circuits Using
Functional Specifications*

Group Art Unit: 234

Examiner V. Trans

April 18, 1989

The Honorable Commissioner of
Patents and Trademarks
Washington, DC 20231

RECEIVED

APR 24 1989

CITATION UNDER 37 CFR 1.97

GROUP 230

Sir:

Attached is a form PTO-1449 listing several documents which may be considered material to the examination of the above identified application. A copy of each cited document is also enclosed. It is requested that these documents be considered by the Examiner and officially made of record in accordance with the provisions of 37 CFR 1.97 and Section 609 of the MPEP.

Respectfully submitted,

A handwritten signature in cursive script, reading "Raymond O. Linker, Jr.".

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561
Our File No. 3868-2
ROLjr:lma

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231, on April 18, 1989

Raymond O. Linker, Jr.
Reg. No. 26,419

Date of Signature

RCL000186

EXHIBIT

43

International Journal of Computer Aided VLSI Design 1, 377-390 (1989)

KBSC: A Knowledge-Based Approach to Automatic Logic Synthesis

HIDEAKI KOBAYASHI

International Chip Corporation

TERUMI SUEHIRO AND MASAHIRO SHINDO

Ricoh Company, Ltd.

A knowledge-based silicon compiler (KBSC) has been jointly developed by Ricoh Company and International Chip Corporation. KBSC provides designers with a front-end graphic interface to automatic logic synthesis. A rule-based expert system synthesizes both data-path and control circuits. KBSC automatically translates an algorithmic description in flowchart form into a logic circuit (netlist) that consists of previously registered cells. ASIC design by KBSC is compared with design by a senior engineer in terms of design time, chip performance, and gate count. An inference engine chip for real-time rule processing is used as a design example. KBSC is also compared with other logic synthesis systems.

ASIC	automatic logic synthesis	flowchart input form
	rule based expert systems	silicon compilation

1 INTRODUCTION

Cost per large-scale integrated (LSI) chip can be defined by development cost D , fabrication cost F , and total production volume N . Therefore, chip cost C can be estimated by $C = D/N + F(N)$, where F is the function of N . If LSI circuits are produced in large volume, the development cost per chip is negligible. LSI circuits of this type include memories and other standard products.

ASICs are produced in small volume to meet customers' needs. In the case of ASICs, however, it becomes important to reduce development cost or design time. The process of ASIC design is divided into functional, logic, and layout stages. To reduce time required for functional and logic design, it is important to

1. Provide an optimum input form to specify an initial chip function.
2. Verify an initial chip function to ensure functional correctness.
3. Automatically translate a verified chip function to a logic circuit or netlist.

Correspondence and requests for reprints should be sent to Hideaki Kobayashi, International Chip Corporation, AT & T Building, 1201 Main St., Suite 2000, Columbia, SC 29201.

■ Manuscript received November 15, 1988; Manuscript revised June 5, 1989.

377

Notice: This material may be
protected by copyright law
(Title 17 U.S. Code).

378 H. Kobayashi, T. Suehiro, and M. Shindo

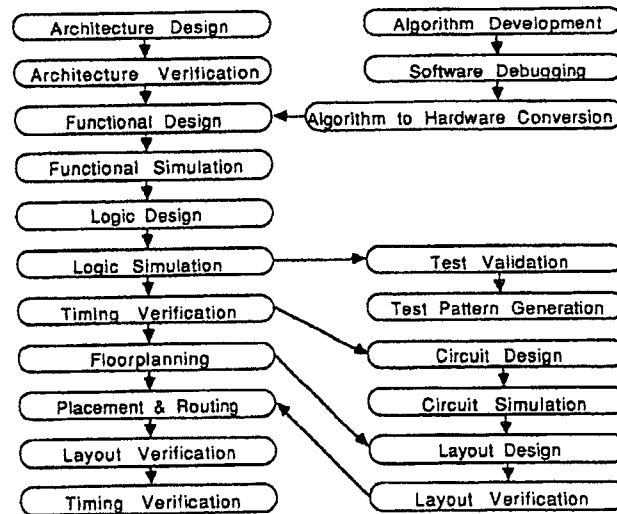


Figure 1. ASIC Development Flow.

Furthermore, to reduce time required for layout design, automatic translation of logic circuit (netlist) information to geometrical mask data is needed.

Many non-commercial logic synthesis systems [1-8] have been developed to translate automatically a behavioral or functional description into a logic circuit or chip layout. The concept of a knowledge-based silicon compiler (KBSC) was introduced in 1987 [9]. KBSC was developed jointly by Ricoh Company and International Chip Corporation (ICC). IF-THEN-type rules have been extracted from expert ASIC designers over several years at Ricoh and ICC, and stored in a knowledge base within KBSC. KBSC and other CAD tools are integrated into a comprehensive CAD system that provides designers with optimum input forms, such as flowcharts, state-transition equations, and functional module diagrams for cell-based design.

2 DESIGN METHODOLOGY

A typical ASIC development flow is shown in Figure 1. First, an entire system function is partitioned into a set of subfunctions (functional blocks). These functional blocks are then defined by various input forms, Figure 2.

1. Boolean expression or truth table form is for combinatorial circuit design.

KBSC 379

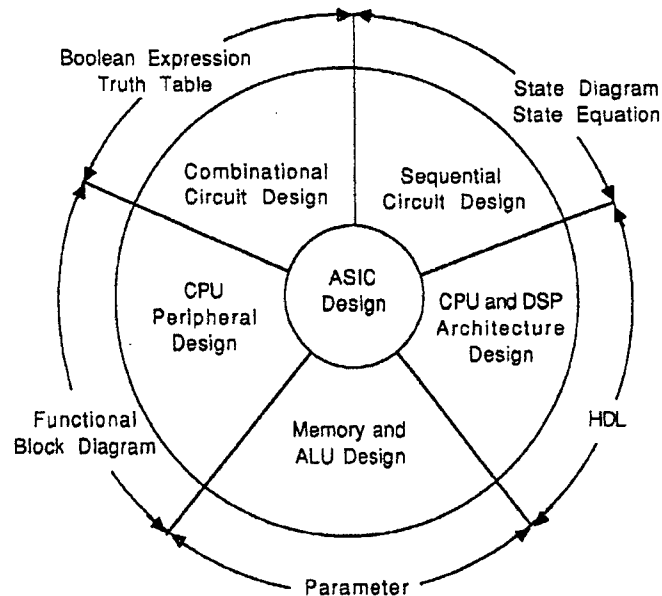


Figure 2. Input Forms for ASIC Design.

2. State-transition diagram or state-equation input form for sequential circuit design.
3. Parameter input form for memory and arithmetic and logic unit (ALU) design with variable address and data lengths.
4. Functional module input form using preregistered cells with bus compatibility for CPU peripheral LSI design.
5. Hardware description language (HDL) input form for CPU and digital signal processing (DSP) architecture design and performance evaluation.

Inputs 1 and 2 above require synthesis and optimization of logic. Logic synthesis and optimization are not required for input 3 since modules are generated by design rules and other constraints. Input 4 utilizes existing logic circuits corresponding to every module. Logic synthesis and optimization are not required since these circuits are already optimized. It is important to select an appropriate input form to design each functional block. The entire system design is completed by interconnecting these functional blocks.

KBSC's flowchart input form (Figure 4) is vital for "system" design that includes both data-path and control logic synthesis. KBSC flowcharts are useful for designers who are familiar with software programming as well as hardware system design. Flowchart-to-netlist conversion is achieved by an expert system where IF-THEN-type rules for logic synthesis are stored in a

380 H. Kobayashi, T. Suehiro, and M. Shindo

knowledge base. These rules are extracted from expert ASIC designers at Ricoh and ICC. KBSC's cell-based design approach provides quick turn-around time and design flexibility. Logic circuits designed by KBSC contain previously registered cells, each with information on its own mask data. Therefore, only placement and routing of these cells need to be performed by layout tools.

Layout design is performed in a hierarchical manner. First, highest level cells (functional blocks) are placed on a chip. An optimum placement is achieved by our floor planner, based on information such as each block area (transistor count), X/Y dimension ratio, and a netlist between blocks. The floor planner computes relative X/Y coordinates, routing areas between blocks, and information about terminal locations in each block.

After initial placement (floor planning), automatic routing takes place. Our automatic place-route software can handle macrocells with four-sided terminals as well as cells with only upper and lower terminals. Savings of 20% or more of the chip area are achieved by this approach compared with our conventional place-route software for two-sided terminals. To achieve 100% routability, routing area is estimated by heuristics. Extra routing area is eliminated by compaction. Layout of lower-level blocks is performed in a similar manner.

A data base stores cells with circuit information and physical mask data. Each cell contains information on logic symbols for schematic capture, models for logic stimulation, terminal names and locations for automatic placement and routing, cell sizes, and physical mask data. These cells are used for automatic logic synthesis as well as schematic capture to design logic circuits.

3 SYSTEM CONFIGURATION

The system configuration of KBSC is shown in Figure 3. The function of each software module in KBSC is explained.

The Flowchart Editor is an interactive functional editor for creation and modification of KBSC flowcharts for design specification entry. An example of a KBSC flowchart is shown in Figure 4. Actions and conditions as well as state transitions are represented by functional macros, which are independent of hardware. These macros are used to define data and numerical operators (e.g., add, subtract, shift, logical AND, logical OR) in each state. A sample list of macros is given in Figure 5.

The Flowchart Simulator is a verification tool for edited flowcharts at the functional level. It guarantees that edited flowcharts represent correct functions. ASIC users easily can operate the Flowchart Simulator to verify functional correctness.

After simulation, flowchart information is separated into actions and conditions. Actions are used for data-path synthesis, and conditions are used for control logic synthesis. Both actions and conditions are described

KBSC 381

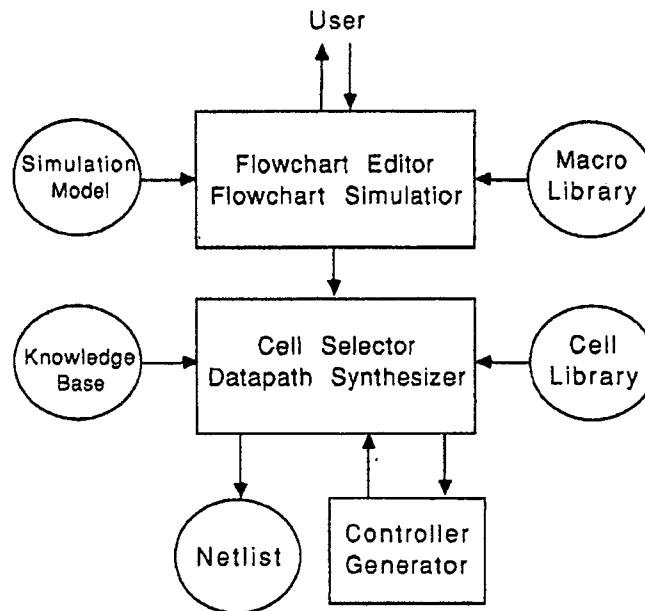


Figure 3. KBSC Configuration.

using the antecedent action form (AAF). An example of AAF is shown in Figure 6.

The Datapath Synthesizer provides automatic logic synthesis and optimization for data-path circuits. Synthesis and optimization of logic are accomplished by applying IF-THEN-type rules. Rules for placement and routing must be prepared for all macros. Example procedures for synthesis and optimization follow:

- Place all macros according to rules for placement.
- Place register blocks for all buses.
- Route between all macros and register blocks. Add control signals to state-transition equations if needed.
- Convert appropriate registers to counters and shifters. Add control signals to state-transition equations if needed.
- Eliminate unnecessary macros and register blocks by classifying all macros and register blocks in terms of function and timing. Insert multiplexers into commonly used macros and register blocks, and add control signals to state-transition equations.
- Connect clock signals. Clock signals for data-path circuits are synthesized separately from clock signals for control circuits. This is to assure that enable signals are stabilized before starting data transfer from data-path circuits.

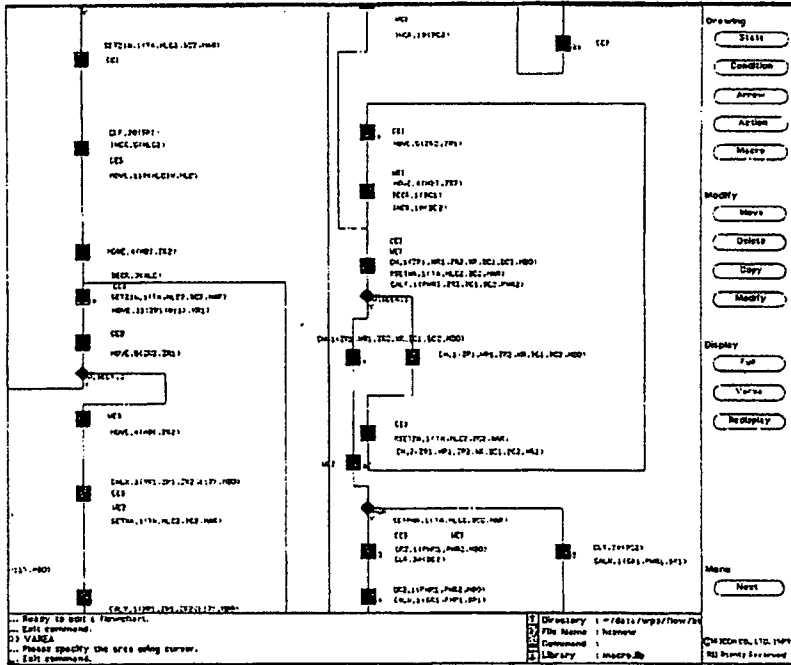


Figure 4. Example of a KBSC Flowchart.

```

*
* CONNECTION MACRO
*
MACRO
( NAME MODE )
( TYPE LOGIC )
( BLOCK INAL )
( BUS / BUS_TYPE IN )
( BUS / BUS_NAME A )
( BUS / BUS_TYPE OUT )
( BUS / BUS_NAME B )
( BUS / BUS_NAME C )
( SIGNAL INAL )
( ENABLE INAL )
( PAIP_MKPO INAL )
CLA
( NAME CLK )
( TYPE GATE )
( BLOCK INAL )
( BUS / BUS_TYPE OUT )
( BUS / BUS_NAME A )
( BUS / BUS_NAME B )
( SIGNAL INAL )
( ENABLE INAL )
( PAIP_MKPO INAL )
*
* COMBINATIONAL MACRO
*
MACRO
( NAME ADD3 )
( TYPE LOGIC )
( BLOCK ADDER )
( BUS / BUS_TYPE IN )
( BUS / BUS_NAME A )
( BUS / BUS_TYPE IN )
( BUS / BUS_NAME B )
( BUS / BUS_NAME C )
( BUS / BUS_TYPE OUT )
( BUS / BUS_NAME D )
( SIGNAL CO )
( ENABLE INAL )
( PAIP_MKPO / NAME SUB3 )
( NAME ADD3 )
( ENABLE / NAME ANDSUB )
( ADD3 ADD )
( SUB3 SUB )

```

Figure 5. Sample List of Macros.

KBSC 383

The Cell Selector uses rules to select existing cells from a library to replace functional cells (without geometrical information) used in data-path synthesis. Example procedures for cell selection follow:

Enter design constraints such as speed, power consumption, and chip area.

Select cells that satisfy entered design constraints.

Eliminate unnecessary circuit portions if a selected cell has unused terminal(s).

The Controller Generator accepts state-transition equations (including modified state-transition equations during data-path synthesis) and auto-

```

KBSC
name htcnew;
data path
    TA<0:3>,      XR<0:7>,      DC1<0:3>,
    DC2<0:3>,      HR1<0:7>,      SR1<0:5>,
    MDI<0:7>,      XR1<0:1>,      MLC<0:2>,
    ZR1<0:7>,      ZR2<0:7>,      MAR<0:7>,
    MDO<0:7>,      MLC2<0:3>,      PHR1<0:7>,
    PHR2<0:7>,    DC1IN<0:3>,    MLCIN<0:2>;
input
    TA,      PHON,      DC1IN,      go,
    MDI,      MLCIN,
    CEB,      WEB,      MDO,      MAR;
reset
    x1:
{
s1 : s2;
s1 : WEB;
s1 : MOVE, 4 (MDI, ZR2);
s1 : DECR, 1 (DC1);
s1 : INCR, 10 (DC2);
s2 : BO, DECR, 1 s3a;
s2 : IBO, DECR, 1 s3;
s2 : CEB;
s2 : WEB;
s2 : CH, 1 (ZR1, HR1, ZR2, XR, DC1, DC2, MDO);
s2 : CALP, 1 (PHR2, ZR2, DC1, DC2, PHR2);
s2 : RSETHA, 1 (TA, MLC2, DC2, MAR);
s4 : s1a;
s4 : CEB;
s4 : RSETZA, 1 (TA, MLC2, DC2, MAR);
s4 : CH, 2 (ZR1, HR1, ZR2, XR, DC1, DC2, HR1);
x1 : go x2;
x1 : lgo x1;
x1 : CLR, 7 (MLC2);
x1 : CLR, 30 (DC2);
x1 : CLR, 2 (ZR1);
x2 : x3;
x2 : CEB;
x2 : SETZ1A, 1 (TA, MLC2, DC2, MAR);
x3 : x4;
x3 : CEB;
x3 : MOVE, 110 (MLCIN, MLC);
x3 : INCR, 5 (MLC2);
x4 : x4a;
x4 : MOVE, 4 (MDI, ZR2);
x4 : CLR, 20 (SR1);
x5 : BO, DECR, 3 x9;
x5 : IBO, DECR, 3 x6;
}
"htcnew.aaf" 129 lines, 2800 characters

```

Figure 6. Example of Antecedent Action Form.

384 H. Kobayashi, T. Suehiro, and M. Shindo

```

=====
PLA Generator Parameter Menu      Ver. 1.3
=====
Selected Design Rule : rccs1d
=====
>> USER INPUT SCREEN <<<
1. Compiled Cell Name : lplm
2. Compiled Cell Data File Name :
=====
>> Compiled Cell Description <<<
1. Number of Input Bits : -----
2. Number of Output Bits : -----
3. Number of Minterms : ---
4. Number of Feedback Loops : --
5. Output Buffer Size : -----
6. Clock Active Edge : -----
7. Cell Size (um) : x x -----
8. Input Data Setup Time (MHz) : -----
9. Clock Width Half (MHz) : -----
10. Output Delay from Clock Active Edge (MHz) : -----
11. Supply Current : -----
=====
Please Input Parameter ( Cell Name & Data File Name )

```

Figure 7. Parameter Menu for PLA Generation.

matically performs synthesis and optimization of logic circuits. Synthesis and optimization can be done in the following two stages.

1. Make a state assignment from state-transition equations and obtain Boolean expressions for circuits that implement state codes and output functions. Example procedures for making a state assignment follow:

Count the total number of states and calculate the number of bits required to represent each state code.

Assign a unique code for each state in order of appearance.

Obtain Boolean expressions from a truth table describing present states, conditions, and next states.

2. Minimize Boolean expressions and generate a parameter menu for PLA generation (Figure 7), or generate a netlist for a logic circuit composed of existing cells. Example procedures for synthesizing a PLA-based system controller follow:

Derive a truth table from Boolean expressions.

Reduce the truth table by applying rules for data compression. Obtain a parameter file for PLA generation.

Example procedures for system controller synthesis using random logic follow:

Convert Boolean expressions to logic using only NAND gates.

Minimize logic by applying rules for optimization.

Convert logic to a circuit using NAND and/or NOR gates with less than or equal to six inputs each.

Minimize logic by applying rules for optimization.

Perform an error check for fan-out number excess and other checks.

A logic circuit (netlist) is generated automatically by combining optimized data-path and control circuits. An example of a logic circuit generated by KBSC is shown in Figure 8.

KBSC 385

4 RULES FOR DATA-PATH SYNTHESIS

Example rule numbers used to synthesize data-paths are shown in Table 1 in execution (firing) order. The number of execution times for each rule is also shown in Table 2. Rule descriptions for data-path synthesis follow:

Rule 10 connects clock and reset terminals to data path circuits.

Rule 100 places an input buffer for an external input bus with a corresponding data length. It also places a register for an internal bus.

Rule 200 places macros (add, sub, and, or). If the macro's first operand is a bus, then it places a block with a corresponding data length. For example, add 0.12 (OP < 16:31 > C1) places a block with 16 bits of data length.

Rule 202 places macros (not, str, sfl). Similar to Rule 200.

Rule 220 places a macro (cmp). Similar to Rule 200.

Rule 230 places output buffers.

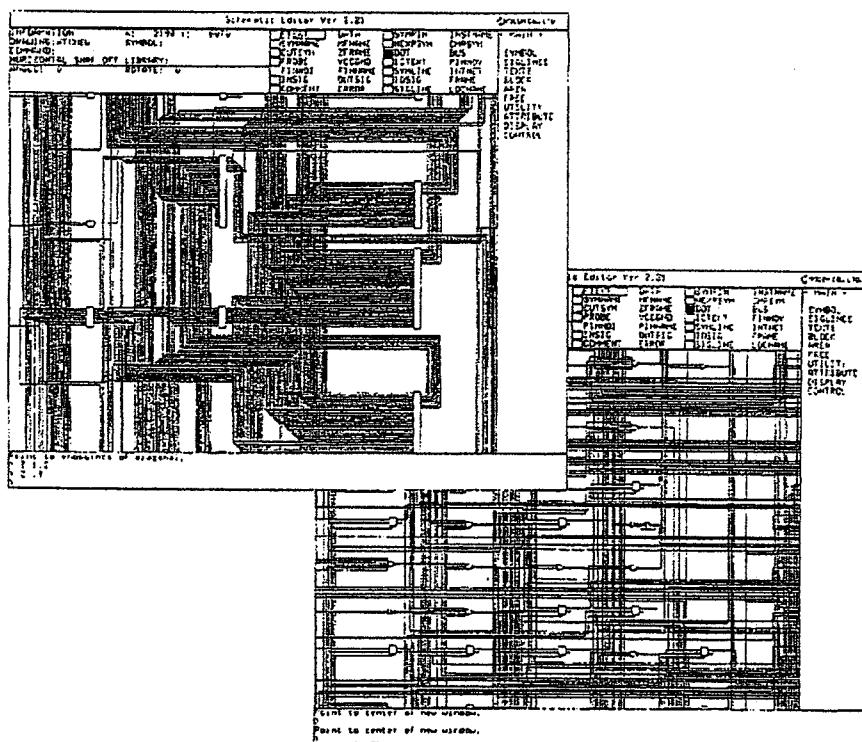


Figure 8. Example of a Logic Circuit Synthesized by KBSC.

386 H. Kobayashi, T. Suehiro, and M. Shindo

Table 1. Rule Firing Sequence for Data-path Synthesis

CONTEXT	BEGIN	RULE 10
CONTEXT	BUSBLOCK	RULE 100
CONTEXT	MACROBLOCK	RULE 200
CONTEXT	MACROBLOCK	RULE 202
CONTEXT	MACROBLOCK	RULE 210
CONTEXT	MACROBLOCK	RULE 220
CONTEXT	MACROBLOCK	RULE 230
CONTEXT	MACROBLOCK	RULE 240
CONTEXT	CONNECTION	RULE 302
CONTEXT	CONNECTION	RULE 308
CONTEXT	CONNECTION	RULE 320
CONTEXT	CONNECTION	RULE 330
CONTEXT	CONNECTION	RULE 332
CONTEXT	CONNECTION	RULE 340
CONTEXT	CONNECTION	RULE 334
CONTEXT	CONNECTION	RULE 310
CONTEXT	CONNECTION	RULE 350
CONTEXT	CONNECTION	RULE 360
CONTEXT	CONNECTION	RULE 370
CONTEXT	CONNECTION	RULE 372
CONTEXT	CONNECTION	RULE 380
CONTEXT	TRANSFORM	RULE 400
CONTEXT	TRANSFORM	RULE 410
CONTEXT	ADDCLOCK	RULE 500
CONTEXT	ADDCLOCK	RULE 510
CONTEXT	ADDCLOCK	RULE 520
CONTEXT	SELECTION	RULE 610
CONTEXT	SELECTION	RULE 620
CONTEXT	SELECTION	RULE 630
CONTEXT	SELECTION	RULE 640
CONTEXT	SELECTION	RULE 650
CONTEXT	SELECTION	RULE 655
CONTEXT	SELECTION	RULE 660
CONTEXT	SELECTION	RULE 665
CONTEXT	SELECTION	RULE 670
CONTEXT	SELECTION	RULE 675
CONTEXT	SELECTION	RULE 680
CONTEXT	SELECTION	RULE 685
CONTEXT	SELECTION	RULE 642
CONTEXT	SELECTION	RULE 644
CONTEXT	SELECTION	RULE 600

Table 2. Number of Executions for Each Rule

RULE 10	TIME 1
RULE 100	TIME 1
RULE 200	TIME 2
RULE 202	TIME 3
RULE 210	TIME 1
RULE 220	TIME 1
RULE 230	TIME 4
RULE 240	TIME 1
RULE 300	TIME 1
RULE 302	TIME 1
RULE 308	TIME 3
RULE 310	TIME 1
RULE 320	TIME 1
RULE 330	TIME 9
RULE 332	TIME 4
RULE 340	TIME 1
RULE 350	TIME 4
RULE 360	TIME 1
RULE 370	TIME 3
RULE 372	TIME 7
RULE 380	TIME 10
RULE 400	TIME 2
RULE 410	TIME 1
RULE 500	TIME 2
RULE 510	TIME 1
RULE 520	TIME 8
RULE 600	TIME 1
RULE 610	TIME 8
RULE 620	TIME 5
RULE 630	TIME 2
RULE 640	TIME 3
RULE 642	TIME 3
RULE 644	TIME 4
RULE 650	TIME 1
RULE 655	TIME 1
RULE 660	TIME 3
RULE 665	TIME 1
RULE 670	TIME 3
RULE 675	TIME 3
RULE 680	TIME 1
RULE 685	TIME 1

KBSC 387

Rule 240 places input buffers.

Rule 210 clears all flags to change mode from placement to routing.

Rule 300 routes macros (add, sub, and, or). It connects a bus register's output terminal, corresponding to the macro's first operand, to the macro's "A" terminal. Similarly, it applies to the macro's second and third operands.

Rule 302 is similar to Rule 300. It applies to the macro's second operand.

A total of 114 rules (42 different types of rules) are applied to synthesize data-paths automatically and to convert them to an equivalent circuit composed of existing cells.

5 RULES FOR CONTROL LOGIC SYNTHESIS

Example rule numbers used to synthesize control logic are shown in Table 3 in execution (firing) order. Rule descriptions for control logic synthesis follow:

Rule 10 constructs a truth table from Boolean expressions.

Rule 100 merges two rows in a truth table with the same combination of inputs into a single row.

Rule 500 checks the fan-out.

Table 3. Rule Firing Sequence for Control Logic Synthesis

[illegible]

388 H. Kobayashi, T. Suehiro, and M. Shindo

A total of 110 rules (18 different types of rules) are applied to synthesize control logic automatically and to convert it to a circuit using existing cells.

6 PERFORMANCE EVALUATION

ASIC design by KBSC was compared with design by a senior engineer in terms of design time, chip performance, and gate count. An inference engine chip for real-time rule processing is used as a design example.

1. *Design time:* KBSC needed only one-fifth of the design time required by a senior engineer. KBSC took approximately three weeks to complete the chip layout, whereas a senior engineer took approximately 15 weeks. Most of the time spent by the engineer was for architecture design.
2. *Chip performance:* The number of cycles to execute an IF-THEN-type rule is defined as a unit parameter. One rule was executed with 13 cycles for the circuit designed by KBSC, whereas the circuit designed by the senior engineer took 8 cycles to execute one rule. This was mainly due to the engineer's experience in designing circuits that can process more than two pieces of data in parallel.
3. *Gate count:* The KBSC circuit required 2,800 gates; the circuit designed by the engineer required 2,500 gates. The difference of 300 gates is due to the lack of rules for eliminating unused gates and optimizing logic circuits.

7 COMPARISON BETWEEN LOGIC SYNTHESIZERS

Logic synthesis systems in References [1-6] are compared with KBSC in terms of flowchart input form and rule-based approach to automatic data-path and control logic synthesis. A technology-oriented register transfer (RT)-level description is used in [1]. Input to KBSC is not an RT-level description, as used in many other approaches. KBSC's input is also not a flowchart like specification of control for typical finite-state machine design. Rules used in KBSC do synthesize both data-path and control circuits from a hardware-independent flowchart description. Output from KBSC is a netlist for automatic placement and routing.

Input to ALERT [2, 3] is an architectural description in Iverson's APL notation with declarations of variables to represent physical devices such as flip-flops and registers. The user of ALERT needs to specify memory size, word length, instruction format, and other hardware design features. In contrast, KBSC's input is a flowchart description with functional macros that are independent of hardware.

Input to the CMU-DA system [4] is an algorithmic description in ISP language. ISP description is translated to the first-level path graph with interconnections of abstract components. It is then mapped to the second level of the data-path graph with selected physical modules.

KBSC 389

The Design Automation Assistant (DAA) [5] is an expert system that uses heuristic rules to synthesize architectural implementation from an algorithmic description with design constraints. Input to DAA is written in ISPS, a description for a Digital Equipment Corporation computer. The DAA produces a hardware network composed of modules, ports, links, and symbolic microcode.

Input to Flamel [6] is a behavioral description in the form of a Pascal program. Flamel's input is associated with a specified bus architecture. Functional blocks such as ALUs, registers, and I/O pads are ordered and placed on buses. Multiplexers are needed to regulate bus traffic at each block's input and output. In contrast, the KBSC approach is not tied to any fixed bus architecture and needs no multiplexers to regulate bus traffic.

SOCRATES [7] uses an expert system to optimize combinatorial logic for a specific target technology. Rules used in SOCRATES optimize gate-level circuits for speed and area in a given technology. A rule replaces a portion of a circuit by a functionally equivalent but more optimal circuit.

8 CONCLUSION

It has been shown that a knowledge-based silicon compiler (KBSC) dramatically reduces time required for functional and logic design. KBSC is clearly distinguished from other logic synthesis systems in terms of its flowchart input form and rule-based approach to automatic data-path and control logic synthesis. More than 10 chips have been designed using KBSC. Applications of these chips include speech synthesis, rule processing, vector-raster conversion, Japanese-character optical character recognition, and printer control.

From these initial results, we believe that KBSC is a useful tool for designers who are not familiar with hardware or circuit design. To improve the current version of KBSC more rules are needed for parallel processing and logic optimization. Most of the problems observed can be solved by simply adding new rules to the knowledge base and modifying rules. The program is written in C language and runs on Sun and other workstations supporting the UNIX operating system and the X windows system.

REFERENCES

- [1] J.A. Darringer, and W.H. Joyner, Jr., "A New Look at Logic Synthesis," *Proc. of 17th Design Automation Conf.*, pp. 543-549, 1980.
- [2] T.D. Friedman, and S.C. Yang, "Methods Used in an Automatic Logic Design Generator (ALERT)," *IEEE Transactions on Computers*, Vol. C-18, No. 7, pp. 593-614, July 1969.
- [3] T.D. Friedman, and S.C. Yang, "Quality of Designs from an Automatic Logic Generator (ALERT)," *Proc. of 7th Design Automation Conf.*, pp. 71-89, 1970.
- [4] A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System—An Example of Automated Data Path

390 H. Kobayashi, T. Suehiro, and M. Shindo

- Design," *Proc. of 16th Design Automation Conf., Las Vegas, NV*, pp. 73-80, 1979.
- [5] T.J. Kowalski, D.J. Geiger, W.H. Wolf, and W. Fichter, "The VLSI Design Automation Assistant: From Algorithms to Silicon," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 33-43, August 1986.
 - [6] H. Trickey, "Flamel: A High-Level Hardware Compiler," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 2, pp. 259-269, March 1987.
 - [7] A.J. deGeus, and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 22-32, August 1986.
 - [8] L. Trevillyan, "An Overview of Logic Synthesis Systems," *Proc. of 24th ACM/IEEE Design Automation Conf.*, Paper 9.1, pp. 166-172, 1987.
 - [9] H. Kobayashi, "KBSC: A Knowledge Based Silicon Compiler—A Future Trend in ASIC Design," Ricoh ASIC Technical Seminar, Tokyo, Japan, October, 1987.

Hideaki Kobayashi (See guest editorial section (p. 355) of this special issue for author's biography).



Terumi Suehiro joined Ricoh Company in 1979 and has been involved in the development of CPU peripheral LSI. Since 1985 he has been developing a Ricoh CAD system for ASICs. He is currently Manager and Senior Engineer of the Semiconductors R & D Center. He received a B.S. degree in applied physics from Nagoya University in 1979.



Masahiro Shindo joined Ricoh Company in 1979 and has been involved in ASIC design, CAD tools, and strategic management. He is currently the Assistant General Manager of the Electronic Devices Division and Director of the Semiconductors R & D Center. Prior to joining Ricoh Company, he was employed by Mitsubishi Electric Corp. and was involved in the development of integrated circuit process technology (micron process, CVD) and devices (DRAM, SRAM, EPROM, LOGIC). He received a B.S. degree in industrial chemical engineering from the University of Ehime in 1963.

EXHIBIT

44

388 H. Kobayashi, T. Suehiro, and M. Shindo

A total of 110 rules (18 different types of rules) are applied to synthesize control logic automatically and to convert it to a circuit using existing cells.

6 PERFORMANCE EVALUATION

ASIC design by KBSC was compared with design by a senior engineer in terms of design time, chip performance, and gate count. An inference engine chip for real-time rule processing is used as a design example.

1. Design time: KBSC needed only one-fifth of the design time required by a senior engineer. KBSC took approximately three weeks to complete the chip layout, whereas a senior engineer took approximately 15 weeks. Most of the time spent by the engineer was for architecture design.
2. Chip performance: The number of cycles to execute an IF-THEN-type rule is defined as a unit parameter. One rule was executed with 13 cycles for the circuit designed by KBSC, whereas the circuit designed by the senior engineer took 8 cycles to execute one rule. This was mainly due to the engineer's experience in designing circuits that can process more than two pieces of data in parallel.
3. Gate count: The KBSC circuit required 2,800 gates; the circuit designed by the engineer required 2,500 gates. The difference of 300 gates is due to the lack of rules for eliminating unused gates and optimizing logic circuits.

7 COMPARISON BETWEEN LOGIC SYNTHESIZERS

Logic synthesis systems in References [1-8] are compared with KBSC in terms of flowchart input form and rule-based approach to automatic data-path and control logic synthesis. A technology-oriented register transfer (RT)-level description is used in [1]. Input to KBSC is not an RT-level description, as used in many other approaches. KBSC's input is also not a flowchart like specification of control for typical finite-state machine design. Rules used in KBSC do synthesize both data-path and control circuits from a hardware-independent flowchart description. Output from KBSC is a netlist for automatic placement and routing.

Input to ALERT [2, 3] is an architectural description in Iverson's APL notation with declarations of variables to represent physical devices such as flip-flops and registers. The user of ALERT needs to specify memory size, word length, instruction format, and other hardware design features. In contrast, KBSC's input is a flowchart description with functional macros that are independent of hardware.

Input to the CMU-DA system [4] is an algorithmic description in ISP language. ISP description is translated to the first-level path graph with interconnections of abstract components. It is then mapped to the second level of the data-path graph with selected physical modules.

The Design Automation Assistant (DAA) [5] is an expert system that uses heuristic rules to synthesize architectural implementation from an algorithmic description with design constraints. Input to DAA is written in ISPS, a description for a Digital Equipment Corporation computer. The DAA produces a hardware network composed of modules, ports, links, and symbolic microcode.

Input to Flamel [6] is a behavioral description in the form of a Pascal program. Flamel's input is associated with a specified bus architecture. Functional blocks such as ALUs, registers, and I/O pads are ordered and placed on buses. Multiplexers are needed to regulate bus traffic at each block's input and output. In contrast, the KBSC approach is not tied to any fixed bus architecture and needs no multiplexers to regulate bus traffic.

SOCRATES [7] uses an expert system to optimize combinatorial logic for a specific target technology. Rules used in SOCRATES optimize gate-level circuits for speed and area in a given technology. A rule replaces a portion of a circuit by a functionally equivalent but more optimal circuit.

8 CONCLUSION

It has been shown that a knowledge-based silicon compiler (KBSC) dramatically reduces time required for functional and logic design. KBSC is clearly distinguished from other logic synthesis systems in terms of its flowchart input form and rule-based approach to automatic data-path and control logic synthesis. More than 10 chips have been designed using KBSC. Applications of these chips include speech synthesis, rule processing, vector-raster conversion, Japanese-character optical character recognition, and printer control.

From these initial results, we believe that KBSC is a useful tool for designers who are not familiar with hardware or circuit design. To improve the current version of KBSC more rules are needed for parallel processing and logic optimization. Most of the problems observed can be solved by simply adding new rules to the knowledge base and modifying rules. The program is written in C language and runs on Sun and other workstations supporting the UNIX operating system and the X windows system.

REFERENCES

- [1] J.A. Darringer, and W.H. Joyner, Jr., "A New Look at Logic Synthesis," *Proc. of 17th Design Automation Conf.*, pp. 543-549, 1980.
- [2] T.D. Friedman, and S.C. Yang, "Methods Used in an Automatic Logic Design Generator (ALERT)," *IEEE Transactions on Computers*, Vol. C-18, No. 7, pp. 583-614, July 1969.
- [3] T.D. Friedman, and S.C. Yang, "Quality of Designs from an Automatic Logic Generator (ALERT)," *Proc. of 7th Design Automation Conf.*, pp. 71-89, 1970.
- [4] A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System—An Example of Automated Data Path

390 H. Kobayashi, T. Suehiro, and M. Shindo

Design." *Proc. of 16th Design Automation Conf.*, Las Vegas, NV, pp. 73-80, 1979.

- [5] T.J. Kowalski, D.J. Geiger, W.H. Wolf, and W. Fichter, "The VLSI Design Automation Assistant: From Algorithms to Silicon," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 33-43, August 1986.
- [6] H. Trickey, "Flam: A High-Level Hardware Compiler," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 2, pp. 259-269, March 1987.
- [7] A.J. deGeus, and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 22-32, August 1986.
- [8] L. Trevillyan, "An Overview of Logic Synthesis Systems," *Proc. of 24th ACM/IEEE Design Automation Conf.*, Paper 9.1, pp. 166-172, 1987.
- [9] H. Kobayashi, "KBSC: A Knowledge Based Silicon Compiler—A Future Trend in ASIC Design," Ricoh ASIC Technical Seminar, Tokyo, Japan, October, 1987.

Hidenori Kobayashi (See guest editorial section (p. 355) of this special issue for author's biography).



Terumi Suehiro joined Ricoh Company in 1979 and has been involved in the development of CPU peripheral LSI. Since 1985 he has been developing a Ricoh CAD system for ASICs. He is currently Manager and Senior Engineer of the Semiconductors R & D Center. He received a B.S. degree in applied physics from Nagoya University in 1979.



Masahiro Shindo joined Ricoh Company in 1979 and has been involved in ASIC design, CAD tools, and strategic management. He is currently the Assistant General Manager of the Electronic Devices Division and Director of the Semiconductors R & D Center. Prior to joining Ricoh Company, he was employed by Mitsubishi Electric Corp. and was involved in the development of integrated circuit process technology (micron process, CVD) and devices (DRAM, SRAM, EPROM, LOGIC). He received a B.S. degree in industrial chemical engineering from the University of Ehime in 1983.

KBSC000872

EXHIBIT

45

Westlaw.

Page 1

37 C.F.R. § 1.56

▷

Effective: [See Text Amendments]

Code of Federal Regulations Currentness
Title 37. Patents, Trademarks, and Copyrights
Chapter I. United States Patent and Trademark
Office, Department of Commerce (Refs &
Annos)
Subchapter A. General
Patents
Part 1. Rules of Practice in Patent Cases
(Refs & Annos)
§ Subpart B. National Processing
Provisions
§ The Application
→§ 1.56 Duty to disclose
information material to
patentability.

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is cancelled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is cancelled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of

any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

(1) prior art cited in search reports of a foreign patent office in a counterpart application, and

(2) the closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

(1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or

(2) It refutes, or is inconsistent with, a position the applicant takes in:

(i) Opposing an argument of unpatentability relied on by the Office, or

(ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

© 2006 Thomson/West. No Claim to Orig. U.S. Govt. Works.

37 C.F.R. § 1.56

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

43953

(1) Each inventor named in the application;

(2) Each attorney or agent who prepares or prosecutes the application; and

Copr. © 2006

Thomson/West

(3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

END OF DOCUMENT

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.

(e) In any continuation-in-part application, the duty under this section includes the duty to disclose to the Office all information known to the person to be material to patentability, as defined in paragraph (b) of this section, which became available between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

[42 FR 5593, Jan. 28, 1977 as amended at 47 FR 21751, May 19, 1982; 48 FR 2710, Jan. 20, 1983; 49 FR 554, Jan. 4, 1984; 50 FR 5171, Feb. 6, 1985; 53 FR 47808, Nov. 28, 1988; 57 FR 2034, Jan. 17, 1992; 65 FR 54666, Sept. 8, 2000]

SOURCE: 24 FR 10332, Dec. 22, 1959; 60 FR 14518, March 17, 1995; 65 FR 14871, March 20, 2000; 65 FR 33455, May 24, 2000; 65 FR 50103, Aug. 16, 2000; 65 FR 56793, Sept. 20, 2000; 65 FR 70490, Nov. 24, 2000, unless otherwise noted.

AUTHORITY: 35 U.S.C. 2(b)(2), unless otherwise noted.

37 C. F. R. § 1.56, 37 CFR § 1.56

Current through August 2, 2006; 71 FR

© 2006 Thomson/West. No Claim to Orig. U.S. Govt. Works.

EXHIBIT

46



UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D. C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
07/143,821			

EXAMINER	
TRANS	
ART UNIT	PAPER NUMBER
234	8

DATE MAILED:

EXAMINER INTERVIEW SUMMARY RECORD

All participants (applicant, applicant's representative, PTO personnel)

- (1) Mr. F. Linker (3) _____
(2) Ex. V. Trans (4) _____

Date of interview Oct 19, 1989

Type: ☐ Telephonic ☒ Personal (copy is given to ☐ applicant ☒ applicant's representative).

Exhibit shown or demonstration conducted: ☐ Yes ☒ No. If yes, brief description: _____

Agreement ☒ was reached with respect to some or all of the claims in question. ☐ was not reached.

Claims discussed: claims 1, 5, 15, 18, 20 and 27

Identification of prior art discussed: Daminger et al. (U.S. Patent 4,903,435)

Description of the general nature of what was agreed to if an agreement was reached, or any other comments: It is agreed that the features "flowchart editor" and "expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit" are patentable distinct from the reference identified above. Thus, applicant's attorney will amend the claims to include these features.
(A fuller description, if necessary, and a copy of the amendments, if available, which the examiner agreed would render the claims allowable must be attached. Also, where no copy of the amendments which would render the claims allowable is available, a summary thereof must be attached.)

Unless the paragraphs below have been checked to indicate to the contrary, A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION IS NOT WAIVED AND MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW (e.g., items 1-7 on the reverse side of this form). If a response to the last Office action has already been filed, then applicant is given one month from this interview date to provide a statement of the substance of the interview.

☐ It is not necessary for applicant to provide a separate record of the substance of the interview.

☐ Since the examiner's interview summary above (including any attachments) reflects a complete response to each of the objections, rejections and requirements that may be present in the last Office action, and since the claims are now allowable, this completed form is considered to fulfill the response requirements of the last Office action.

[Signature]
Examiner's Signature

PTOL-413 (REV. 1-84)

ORIGINAL FOR INSERTION IN FRONT BOARD FILE OR FILE WRAPPER

RCL000228

EXHIBIT

47

Flamel: A High-Level Hardware Compiler

HOWARD TRICKEY, MEMBER, IEEE

Abstract—This paper describes the design and implementation of a high-level hardware compiler called *Flamel*. Ordinary Pascal programs are used to define the behavior required of the hardware. *Flamel* undertakes to find parallelism in the program, so it can produce a fast-running implementation that meets a user-specified cost bound.

A number of program transformations create sections of code with more parallel computations than the original program has. A novel feature of *Flamel* is a method for organizing the search for the transformations that best satisfy the goal. Another new algorithm is one for "expression height reduction": rewriting an ensemble of expressions using algebraic properties in order to compute the expressions faster.

An implementation of *Flamel* has been completed. The output is a description of a datapath and a controller, and at a sufficient level of detail so that good area and execution time figures can be estimated. On a series of tests, *Flamel* produces implementations of programs that would run 22 to 200 times faster than an MC68000 running the same programs, if the clock cycles were the same. The tests also show that a wide range of time-area tradeoffs are produced by varying the area constraint.

I. INTRODUCTION

MUCH OF THE early work in hardware synthesis took hardware descriptions that were mostly *structural*, meaning that the input described the architecture of the required system and the synthesis systems instantiated that architecture in some technology. More recently, it has become feasible to build compilers that accept *behavioral* descriptions, where the input simply describes what the required system needs to do. Such compilers need to choose an architecture; after that, the techniques developed for the structural synthesizers can be used. This paper describes a behavioral hardware compiler called *Flamel*.

A typical way of describing behavior is to write a program in an ordinary computer language, with some convention as to how to communicate with the "outside world." Roughly speaking, the required system needs to act the same way across the communication channel that the program does. The hardware compiler can rearrange the program and have calculations done in parallel, so long as the proper values are written to the outside world at the proper times. Some examples of compilers that operate this way are: the CMU-DA project [1], particularly the Design Automation Assistant [2], [3] portion; Arsenic [4];

Manuscript received February 16, 1986; revised November 12, 1986. This work was done while the author was at Stanford University. Supported by DARPA contract N00039-83-C-0136.

The author is with AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

IEEE Log Number 8612997.

"After a medieval alchemist said by some to have achieved the "Great Work" of gold-making.

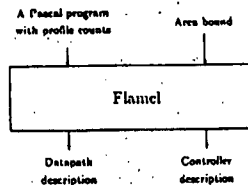


Fig. 1. Flamel's operation.

the USC Design Automation project [5]; the AT&T Bell Labs VLSI Design Automation Assistant [6]; and SC [7]. There are some similarities between *Flamel* and these other systems. The thing that distinguishes *Flamel* from the others is extensive modification of the input program, together with an algorithm for deciding which modifications yield the fastest-executing chips while meeting a user-supplied cost constraint.

An overall picture of *Flamel*'s operation is shown in Fig. 1. The user provides a Pascal program together with execution frequency counts for a typical execution of the input program. The other user input is a number saying roughly how much hardware is allowed. The output is a design for hardware that will perform the same function as the Pascal program.

II. THE COMPILATION PROBLEM

The problem that *Flamel* attempts to solve is:

Given a program P , find a hardware implementation with the same external behavior as P . The implementation should minimize the execution time of the implementation running on typical data, while meeting a user-supplied constraint on the cost of the hardware implementation.

This section will define *hardware implementation* and *external behavior*, and explain how *Flamel* estimates execution time and hardware cost.

A. Hardware Model

The general model for a circuit produced by *Flamel* is that of a synchronous digital machine consisting of a *datapath* and a *controller*. The datapath consists of *functional units* (ALU's, adders, registers, I/O pads, etc.) interconnected by busses. Functional units source, sink, hold, and operate on data words, where a word contains some number of bits. A given bus may receive data from and give data to several functional units. This means that

EXHIBIT

48

taneously. It remains to be answered how much of this type of parallelism can be expected, but the results reported elsewhere [28], [29] show that at least a 5–10 times speedup can be expected from scientific programs, and the results of this work show that a 2–8 times speedup can be expected from general-purpose software.

Some of the contributions of this work are as follows.

- It has been shown that one easy-to-write behavior specification can generate a wide range of time/area trade-off implementations, with a bare minimum of effort required from the user. Up to now, general-purpose digital system design has been regarded as a domain where an expert is needed to either choose the architecture or at least to guide the choice.

- An adequate set of global (block-level) and local (DAG-level) transformations has been identified, along with a controlled method for applying them. It is easy to suggest optimizations to apply during silicon compilation—after all, most of them have long been known to the software compiler community. Others have done this, but none have given feasible methods for deciding which to apply and when. For instance, Kowalski's system incorporates a number of global optimizations, but the user has to decide by hand which to apply and when [2]. Flamel's dacon-choosing procedure is a step in that direction. Also, the optimizations described here have all been found to be useful in practice, since they were added on an as-needed basis.

- This is the first work to do extensive global optimizations at all; yet the results show that the global optimizations are crucial for extracting much parallelism from ordinary programs. (Fisher's trace scheduling [29] achieves a similar effect by scheduling at a global level; but he intends it for a fixed architecture).

- A height-reduction technique has been developed to work on DAG's rather than simply trees. The combination of the height-reduction technique and the other transforms yields results that subsume a number of specialized techniques used in vectorizing compilers.

- The prototype implementation revealed useful statistics on the relative cost and effectiveness of the proposed optimization techniques. For instance, it showed what size and speed might be expected from a Flamel-designed chip for various pieces of code. Also, the fact that Flamel runs reasonably quickly shows that a moderate amount of search (in dacon-choosing) is not out of the question.

There is a lot of room for improvement in Flamel. Trickey [15] discusses some ideas in a number of areas that deserve further investigation: 1) more than one procedure in the input program; 2) handling the timing of communication with the outside world; 3) pipelining, both at the operation level and the block level; 4) special implementations for arrays; 5) incorporating controller area into the costs.

REFERENCES

- [1] A. C. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Lieve, and J. Kim, "The CMU design automation system," in *ACM IEEE 16th Design Automation Conf. Proc.* (San Diego), 1979, pp. 73–80.
- [2] T. J. Kowalski, "The VLSI design automation assistant: A knowledge-based expert system," Ph.D. thesis, Carnegie Mellon Univ., 1984. Available as CMU Technical Report CMUCAD-84-29.
- [3] T. J. Kowalski and D. E. Thomas, "The VLSI design automation assistant: Prototype system," in *ACM IEEE 20th Design Automation Conf. Proc.* (Miami), 1983, pp. 479–483.
- [4] K. Palem, D. S. Fussell, and A. J. Welch, "High-level optimization in a silicon compiler," Tech. Rep. TR-215, Dept. of Computer Sciences, University of Texas, Austin, TX, 1982.
- [5] D. Knapp, J. Granacki, and A. C. Parker, "An expert synthesis system," in *Proc. of the Int. Conf. on Computer Aided Design*, ACM and IEEE, 1983, pp. 419–424.
- [6] T. J. Kowalski, D. J. Geiger, W. H. Wolf, and W. Pichtner, "The VLSI design automation assistant: From algorithms to silicon," *IEEE Design and Test*, pp. 33–43, Aug. 1985.
- [7] P. E. Agre, "Designing a high-level silicon compiler," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers* (Port Chester, NY), 1983, p. 413.
- [8] J. M. Siskind, J. R. Southard, and K. W. Crouch, "Generating custom high performance VLSI designs from succinct algorithmic descriptions," in *Proc. Conf. on Advanced Research in VLSI*, (MIT), Jan. 1982, pp. 28–39.
- [9] H. E. Shrobe, "The data path generator," in *Proc. Conf. on Advanced Research in VLSI*, (MIT), Jan. 1982, pp. 175–181.
- [10] J. L. Hennessey, "SLIM: A simulation and implementation language for VLSI microcode," *Lambda*, pp. 20–28, Apr. 1981.
- [11] A. R. Kartlin, H. Trickey, and J. D. Ullman, "Experience with a regular expression compiler," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers* (Port Chester, NY), 1983, pp. 656–663.
- [12] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*. Reading, MA: Addison-Wesley, 1977.
- [13] M. C. McFarland, "The VT: A database for automated digital design," Tech. Rep. DRC-01-4-80, Design Research Center, Carnegie Mellon University, 1978.
- [14] G. S. Tjaden and M. J. Flynn, "Detection and parallel execution of independent instructions," *IEEE Trans. Comput.*, vol. C-19, no. 10, pp. 889–895, 1970.
- [15] H. Trickey, "Compiling Pascal programs into silicon," Ph.D. thesis, Stanford Univ., July 1985. Stanford Computer Science Report STAN-CS-85-1059.
- [16] D. J. Kuck, *The Structure of Computers and Computations*, vol. 1. New York: Wiley, 1978.
- [17] P. M. Kogge and H. S. Stone, "A parallel algorithm for efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, pp. 786–792, Aug. 1973.
- [18] U. Banerjee, D. Gajski, and D. J. Kuck, "Array machine control units for loops containing IFs," in *Proc. 1980 IEEE Int. Conf. on Parallel Processing*, 1980, pp. 23–36.
- [19] C. Y. Hitchcock III and D. E. Thomas, "A method of automatic data path synthesis," in *ACM IEEE 20th Design Automation Conf. Proc.* (Miami), 1983, pp. 484–489.
- [20] J. R. Southard, "MacPitts: An approach to silicon compilation," *Computer*, vol. 16, pp. 74–82, Dec. 1983.
- [21] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
- [22] P. Chow, "A portable machine-independent global optimizer—Design and measurements," Ph.D. thesis, Stanford Univ., Dec. 1983.
- [23] S. Przybylski, T. Gross, J. Hennessey, N. Jouppi, and C. Rowen, "Organization and VLSI implementation of MIPS," *J. of VLSI and Computer Systems*, vol. 1, Spring 1984. Available as Tech. Rep. 83-259, CSL, Stanford.
- [24] B. W. Kernighan and P. J. Plauger, *Software Tools in Pascal*. Reading, MA: Addison-Wesley, 1981.
- [25] J. Newkirk, R. Mathews, J. Redford, and C. Burns, "Stanford nMOS cell library," Tech. Rep. 001, Information Systems Laboratory, Stanford Univ., 1981.
- [26] D. D. Gajski, D. A. Padua, D. J. Kuck, and R. H. Kuhn, "A second opinion on data flow machines," *Computer*, vol. 15, pp. 58–69, Feb. 1982.
- [27] J. A. Fisher, J. R. Ellis, J. C. Ruttenberg, and A. Nicolau, "Parallel processing: A smart compiler and a dumb machine," in *Proc. ACM SIGPLAN Symp. on Compiler Construction* (Montreal, Canada), 1984, pp. 37–47.

EXHIBIT

49

Westlaw.

Page 1

37 C.F.R. § 1.97

▷

Effective: [See Text Amendments]

Code of Federal Regulations Currentness
 Title 37. Patents, Trademarks, and Copyrights
 Chapter I. United States Patent and Trademark
 Office, Department of Commerce (Refs &
 Annos)
 Subchapter A. General
 Patents
 Part 1. Rules of Practice in Patent Cases
 (Refs & Annos)
 ¶ Subpart B. National Processing
 Provisions
 ¶ Information Disclosure Statement
 → § 1.97 Filing of information
 disclosure statement.

(a) In order for an applicant for a patent or for a reissue of a patent to have an information disclosure statement in compliance with § 1.98 considered by the Office during the pendency of the application, the information disclosure statement must satisfy one of paragraphs (b), (c), or (d) of this section.

(b) An information disclosure statement shall be considered by the Office if filed by the applicant within any one of the following time periods:

- (1) Within three months of the filing date of a national application other than a continued prosecution application under § 1.53(d);
- (2) Within three months of the date of entry of the national stage as set forth in § 1.491 in an international application;
- (3) Before the mailing of a first Office action on the merits; or
- (4) Before the mailing of a first Office action after the filing of a request for continued examination under § 1.114.

(c) An information disclosure statement shall be

considered by the Office if filed after the period specified in paragraph (b) of this section, provided that the information disclosure statement is filed before the mailing date of any of a final action under § 1.113, a notice of allowance under § 1.311, or an action that otherwise closes prosecution in the application, and it is accompanied by one of:

- (1) The statement specified in paragraph (e) of this section; or
- (2) The fee set forth in § 1.17(p).

(d) An information disclosure statement shall be considered by the Office if filed by the applicant after the period specified in paragraph (c) of this section, provided that the information disclosure statement is filed on or before payment of the issue fee and is accompanied by:

- (1) The statement specified in paragraph (e) of this section; and
- (2) The fee set forth in § 1.17(p).

(e) A statement under this section must state either:

- (1) That each item of information contained in the information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of the information disclosure statement; or
- (2) That no item of information contained in the information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in the information disclosure statement was known to any individual designated in § 1.56(c) more than three months prior to the filing of the information disclosure statement.

© 2006 Thomson/West. No Claim to Orig. U.S. Govt. Works.

37 C.F.R. § 1.97

(f) No extensions of time for filing an information disclosure statement are permitted under § 1.136. If a bona fide attempt is made to comply with § 1.98, but part of the required content is inadvertently omitted, additional time may be given to enable full compliance.

(g) An information disclosure statement filed in accordance with this section shall not be construed as a representation that a search has been made.

(h) The filing of an information disclosure statement shall not be construed to be an admission that the information cited in the statement is, or is considered to be, material to patentability as defined in § 1.56(b).

(i) If an information disclosure statement does not comply with either this section or § 1.98, it will be placed in the file but will not be considered by the Office.

[42 FR 5594, Jan. 28, 1977, as amended at 48 FR 2712, Jan. 20, 1983; 57 FR 2034, Jan. 17, 1992; 59 FR 32658, June 24, 1994; 60 FR 20226, April 25, 1995; 61 FR 42805, Aug. 19, 1996; 62 FR 53190, Oct. 10, 1997; 65 FR 14872, March 20, 2000; 65 FR 50103, Aug. 16, 2000; 65 FR 54670, Sept. 8, 2000]

SOURCE: 24 FR 10332, Dec. 22, 1959; 60 FR 14518, March 17, 1995; 65 FR 14871, March 20, 2000; 65 FR 33455, May 24, 2000; 65 FR 50103, Aug. 16, 2000; 65 FR 56793, Sept. 20, 2000; 65 FR 70490, Nov. 24, 2000, unless otherwise noted.

AUTHORITY: 35 U.S.C. 2(b)(2), unless otherwise noted.

37 C. F. R. § 1.97, 37 CFR § 1.97

Current through August 2, 2006; 71 FR 43953

Copr. © 2006

Thomson/West

END OF DOCUMENT